special

Valentin Christiaens

Sep 19, 2023

GETTING STARTED

Recent technological progress in high-contrast imaging has allowed the spectral characterization of directly imaged giant planet and brown dwarf companions at ever shorter angular separation from their host stars, hence opening a new avenue to study their formation, evolution, and composition.

In this context, `special` is a Python package for the SPEctral Characterization of directly ImAged Low-mass companions. While some tools are specific to the characterisation of low-mass (M, L, T) dwarfs down to giant planets at optical/IR wavelengths, the main routines of `special` (MCMC and nested samplers) can also be used in a more general way for the characterisation of any type of object with a measured spectrum, provided a relevant input model grid, regardless of the observational method used to obtain the spectrum (direct imaging or not) and regardless of the format of the spectra (multi-band photometry, low-resolution or medium-resolution spectrum, or a combination thereof).

CHAPTER

ONE

WHAT MAKES IT *SPECIAL*?

The `special` package provides a number of tools for the analysis of spectra from any (sub)stellar object, regardless of the observational method used to obtain the spectra (direct imaging or not) and the format of the spectra (multi-band photometry, low-resolution or medium-resolution spectrum, or a combination thereof). That being said, the main routines in the package (e.g. Bayesian retrieval of model parameters though MCMC or nested samplers, or best-fit template search) can also be applied to the spectrum of any object, provided a relevant grid of models or library of templates for the fit.

The main available features of the package are listed below:

- calculation of the spectral correlation between channels of an IFS datacube (relevant to directly imaged companions with an IFS, where the uncertainty reflects spectrally correlated residual speckle noise);
- calculation of empirical spectral indices for MLT-dwarfs;
- fitting of input spectra to either photo-/atmospheric model grids or a blackbody model, including additional parameters such as (extra) black body component(s), extinction and total-to-selective extinction ratio;
- using either MCMC (emcee) or nested (nestle or UltraNest) samplers to infer posterior distributions on spectral model parameters in a Bayesian framework;
- searching for the best-fit template spectrum within a given template library, with up to two free parameters (relative flux scaling and extinction).

The MCMC and nested sampler routines have been adapted to:

- be flexible, as they are usable on any grid of models provided by the user (along with a snippet function specifying how to read the format of the input files);
- sample the effect of (additional) blackbody components;
- sample the effect of extinction (AV);
- sample different extinction laws than ISM (parametrised using the total-to-selective extinction ratio RV);
- sample a list of potential emission lines;
- accept either uniform or Gaussian priors for each model parameter;
- accept a prior on the mass of the object (if surface gravity is one of the model parameters, and for the MCMC sampler only);
- consider convolution with the line spread function, photometric filters transmission and/or resampling of the model for consistency with the input spectrum - in particular convolution and resampling are done in two consecutive steps, and multiple resolving powers can be provided as input;
- use a log-likelihood expression that can include i) spectral correlation between measurements of adjacent channels of a given instrument, and ii) additional weights that are proportional to the relative spectral bandwidth of each measurement, in case these are obtained from different instruments (e.g. photometry+spectroscopy).

`special` was originally developed for the specific need of characterizing (sub)stellar companion CrA-9b/B at a time when no other public package achieving this task was available (Christiaens et al. (2021)). First implemented as `specfit`, a former module of the `VIP` package, it then underwent significant expansion, with the current package now detailed in the following JOSS publication Christiaens et al. (2022) (preferred citation if you use this package).

CHAPTER

TWO

TL;DR SETUP GUIDE

```
$ pip install special
```

CHAPTER

THREE

INSTALLATION AND DEPENDENCIES

The benefits of using a Python package manager (distribution), such as (ana)conda or Canopy, are many. Mainly, it brings easy and robust package management and avoids messing up with your system's default python. An alternative is to use package managers like apt-get for Ubuntu or Homebrew/MacPorts/Fink for macOS. We recommend using Miniconda.

`special` depends on existing packages from the Python ecosystem, such as `numpy`, `scipy`, `matplotlib`, `pandas` and `astropy`. There are different ways of installing `special` suitable for different scenarios.

3.1 Using pip

The easiest way to install `special` is through the Python Package Index, aka PyPI, with the `pip` package manager. Simply run:

```
$ pip install special
```

With `pip` you can easily uninstall, upgrade or install a specific version of `special`. For upgrading the package run:

```
$ pip install --upgrade special
```

Alternatively, you can use `pip install` and point to the GitHub repo:

```
$ pip install git+https://github.com/VChristiaens/special.git
```

3.2 Using the setup.py file

You can download `special` from its GitHub repository as a zip file. A `setup.py` file (setuptools) is included in the root folder of `special`. Enter the package's root folder and run:

```
$ python setup.py install
```

3.3 Using Git

If you plan to contribute or experiment with the code you need to make a fork of the repository (click on the fork button in the top right corner) and clone it:

```
$ git clone https://github.com/<replace-by-your-username>/special.git
```

If you do not create a fork, you can still benefit from the `git` syncing functionalities by cloning the repository (but will not be able to contribute):

```
$ git clone https://github.com/VChristiaens/special.git
```

Before installing the package, it is highly recommended to create a dedicated conda environment to not mess up with the package versions in your base environment. This can be done easily with (replace spec_env by the name you want for your environment):

```
$ conda create -n spec_env python=3.9 ipython
```

Note: installing ipython while creating the environment with the above line will avoid a commonly reported issue which stems from trying to import `special` from within a base python2.7 ipython console.

To install `special`, simply cd into the special directory and run the setup file in 'develop' mode:

```
$ cd special
$ python setup.py develop
```

If cloned from your fork, make sure to link your special directory to the upstream source, to be able to easily update your local copy when a new version comes out or a bug is fixed:

```
$ git add remote upstream https://github.com/VChristiaens/special.git
```

3.4 Loading *special*

Finally, start Python or IPython and check that you are able to import `special`:

```
import special
```

Now you can start characterizing exoplanets and other (sub)stellar objects!

CHAPTER

FOUR

SPECIAL TUTORIAL

Written by: *Valentin Christiaens*.

Last update: *2022/08/04*

IMPORTANT: to run this jupyter notebook tutorial, you will have to download the `special-extras` repository, and keep the `utils.py` file (containing utility routines specific to the dataset used in the notebook) in the same folder as the notebook. Alternatively, you can execute this notebook on Binder (in the tutorials directory).

4.1 Table of contents

Let's start by importing a couple of useful packages. It is necessary to have both the `utils.py` file located in the same directory as this notebook, and the datasets in a folder at the same level as the parent directory to this notebook. This is the default if you download the GitHub repository and run the notebook from within your local copy of the repository, or if you use Binder.

[1]:
```
import special
special.__version__
```

[1]:
```
'0.1.5'
```

[2]:
```
%matplotlib inline
from matplotlib import pyplot as plt
import numpy as np
from pathlib import Path
```

CHAPTER

FIVE

1. LOADING THE DATA

The data used in this tutorial correspond to (i) the 2MASS J19005804-3645048 b/B (aka. CrA-9 b/B) spectrum acquired from VLT/SPHERE-IFS, VLT/SPHERE-IRDIS and VLT/NACO high-contrast imaging data, and (ii) the SPHERE-IFS PSF-subtracted datacube, presented in Christiaens et al. (2021). You can find them in the 'dataset' folder of the `special_extras` repository:

- `CrA-9b_spectrum.fits`: the measured spectrum of (sub)stellar companion CrA-9 b/B, as derived using the negative fake companion technique (Wertz et al. 2017; i.e. finding the negative flux that minimizes the absolute residuals at the location of the companion) in each IFS spectral channel and in each broad-band filter image obtained with IRDIS and NACO;
- `CrA-9_2019_IFS_pca_annulus.fits`: the post-processed Integral Field Spectrograph (IFS) datacube on CrA-9, where the stellar halo and speckles have been modeled by principal component analysis (PCA; Amara & Quanz 2012) and removed in each of the 39 spectral channels.

Let's define our input path:

[3]:
```
par_path = Path('__file__').parent
rel_path = '../datasets/'
datpath = str((par_path / rel_path).resolve())+'/'
```

In the post-processed IFS cube, the emission from the central star has been modeled and subtracted using the ADI strategy in each spectral channel. The post-processing was performed using principal component analysis (PCA) applied in a single annulus. The annulus was chosen to encompass the faint (sub)stellar companion CrA-9b/B, which can be seen towards the left of the images.

Let's first inspect the IFS cube using the `info_fits` utility implemented in the `fits` module:

[4]:
```
from special.fits import info_fits
info_fits(datpath+'CrA-9_2019_IFS_pca_annulus.fits')
```

```
Filename: /Users/valentin/GitHub/special_extras/datasets/CrA-9_2019_IFS_pca_annulus.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU       7   (291, 291, 39)   float64
```

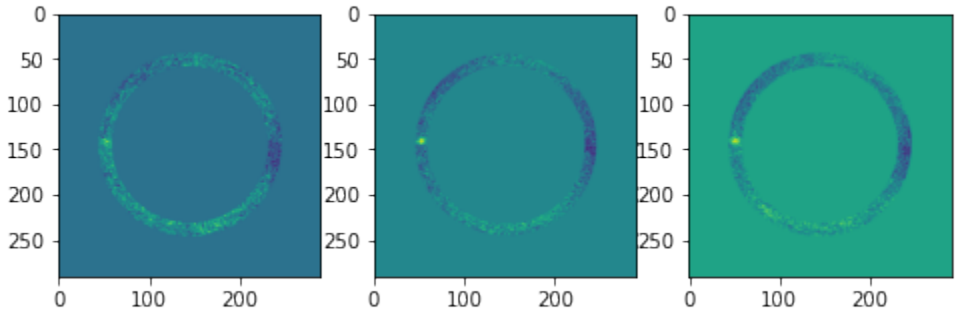
The fits file contains an IFS datacube of 39 images with wavelengths spanning 0.95 to 1.68 μm. Each image is 291x291 px across. Let's now open it and check a few spectral channels.

[5]:
```
from special.fits import open_fits
ifs_cube = open_fits(datpath+'CrA-9_2019_IFS_pca_annulus.fits')
```

```
Fits HDU-0 data successfully loaded. Data shape: (39, 291, 291)
```

[6]:
```
from special.config import figsize
fig, axes = plt.subplots(1,3,figsize=figsize)
axes[0].imshow(ifs_cube[1])
axes[1].imshow(ifs_cube[19])
axes[2].imshow(ifs_cube[-1])
```

[6]:
```
<matplotlib.image.AxesImage at 0x7fd0d0a9d850>
```



A point source can be seen towards the left of the image. This is CrA-9 b/B, a faint companion to the pre-main sequence star CrA-9, whose nature is still debated (Christiaens et al. 2021). In addition to this IFS datacube obtained with VLT instrument SPHERE/IFS, three additional detections of the companion were obtained with VLT imagers SPHERE/IRDIS at 2.11 μm and 2.25μm, and NACO at 3.8 μm. The full measured spectrum contains thus 42 points:

[7]:
```
info_fits(datpath+'CrA-9b_spectrum.fits')
```

```
Filename: /Users/valentin/GitHub/special_extras/datasets/CrA-9b_spectrum.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU       6   (42, 4)   float32
```

The 4 dimensions correspond to:

- the wavelength of each measurement (in μm);
- the channel width (or FWHM of the broad band filter used in the case or IRDIS and NACO);
- the flux of the companion;
- the error on the flux.

[8]:
```
spectrum_b = open_fits(datpath+'CrA-9b_spectrum.fits')

lbda = spectrum_b[0]
dlbda = spectrum_b[1]
spec = spectrum_b[2]
spec_err = spectrum_b[3]
```

```
Fits HDU-0 data successfully loaded. Data shape: (4, 42)
```

Let's visualize the measured spectrum in λF_λ units:

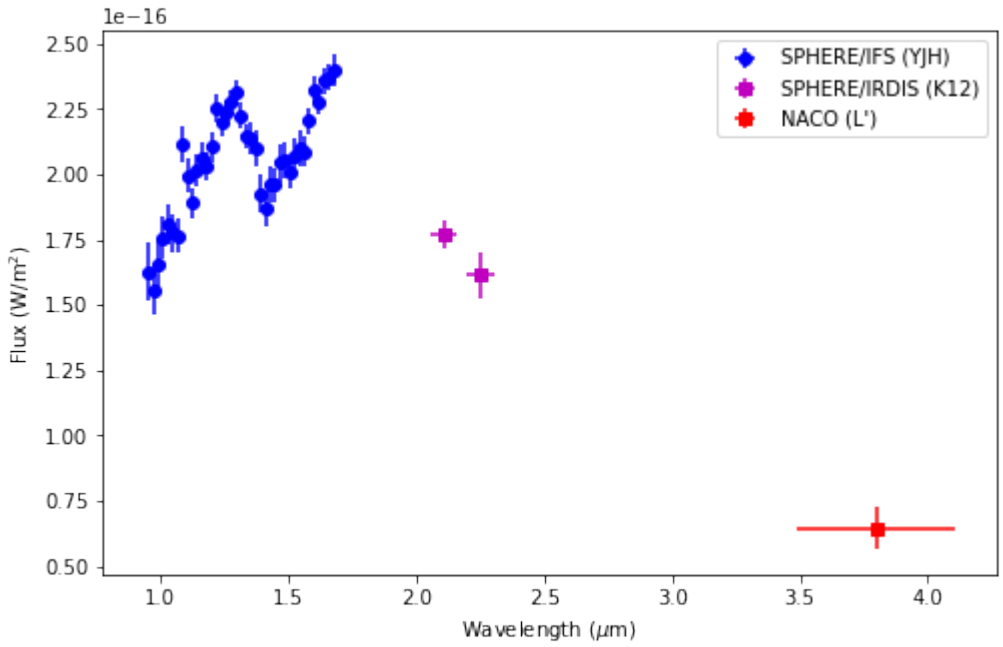
[9]:
```
fig = plt.figure(figsize=figsize)
plt.errorbar(lbda[:-3], lbda[:-3]*spec[:-3], lbda[:-3]*spec_err[:-3], dlbda[:-3], 'bo',
             label = 'SPHERE/IFS (YJH)')
plt.errorbar(lbda[-3:-1], lbda[-3:-1]*spec[-3:-1], lbda[-3:-1]*spec_err[-3:-1], dlbda[-3:
↪-1], 'ms',
             label = 'SPHERE/IRDIS (K12)')
```

(continues on next page)

(continued from previous page)

```
plt.errorbar(lbda[-1], lbda[-1]*spec[-1], lbda[-1]*spec_err[-1], dlbda[-1], 'rs',
             label = "NACO (L')")
plt.xlabel(r"Wavelength ($\mu$m)")
plt.ylabel(r"Flux (W/m$^2$)")
plt.legend()
```

[9]:
```
<matplotlib.legend.Legend at 0x7fd0d0d2c7c0>
```



Go to the top

CHAPTER

SIX

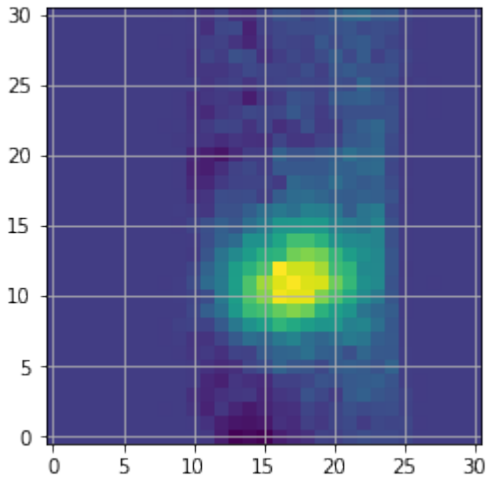
2. SPECTRAL CORRELATION MATRIX

6.1 2.1 Estimating the spectral correlation between IFS channels

Before fitting different models to the spectrum, let's first estimate the spectral correlation between the IFS channels. By design of the IFS, the flux at a given wavelength leaks on/overlaps with neighbouring channels. These can therefore not be considered as independent measurements. The spectral correlation matrix has therefore to be considered when fitting a given IFS spectrum. Not taking it into account can indeed lead to significant biases, and in particular to erroneous results in terms of best-fit model parameters (see e.g. Greco & Brandt 2016).

Estimating the spectral correlation can be done easily in `special` using the `spectral_correlation` routine, which follows the method recommended in Greco & Brandt (2016). For that let's first determine the location of the companion in the IFS images, by zooming on the median image of the cube:

[10]:
```
x_i = 35
y_i = 130
sz = 31
plt.imshow(np.median(ifs_cube,axis=0)[y_i:y_i+sz,x_i:x_i+sz])
plt.gca().invert_yaxis()
plt.grid()
```



Based on the above inset, we can set the approximate coordinates of the companion, and infer its radial separation:

[11]:
```
pl_xy = ((x_i+17,y_i+11),) # set pl_xy as a tuple of 2-element tuples - each tuple␣
↪corresponds to a different companion
```

(continues on next page)

(continued from previous page)

```
cy = (ifs_cube.shape[1]/2.)-0.5
cx = (ifs_cube.shape[2]/2.)-0.5
r0 = np.sqrt((pl_xy[0][0]-cx)**2+(pl_xy[0][1]-cy)**2)
r0
```

[11]:
```
93.08598175880189
```

The spectral correlation depends on the radial separation to the central star. The algorithm implemented in `special` can therefore compute it in concentric annuli between given inner and outer radii (by default spanning the whole field). Here since we are only interested in the spectral correlation at the radial separation of the companion, let's set the annular width, inner and outer radii such that they only encompass the companion radial separation:

[12]:
```
awidth=2
r_in = int(r0-awidth/2)
r_out = int(r0+awidth/2)
```

Furthermore, we mask the area around the companion to not bias the spectral correlation estimate with companion flux. We set the mask radius to be 4 times the FWHM, with a FWHM set to 4px:

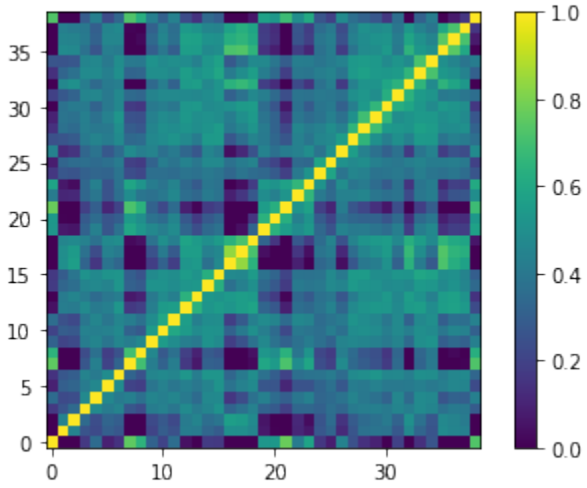
[13]:
```
mask_r=4
fwhm=4
```

Now, we can estimate the spectral correlation at the radial separation of CrA-9b/B:

[14]:
```
from special.spec_corr import spectral_correlation
sp_corr = spectral_correlation(ifs_cube, awidth=awidth, r_in=r_in, r_out=r_out, pl_xy=pl_
↪xy,
                               mask_r=mask_r, fwhm=fwhm)
sp_corr = sp_corr[0]
```

Let's visualize it:

[15]:
```
plt.imshow(sp_corr)
#plt.savefig("spectral_corr_matrix.pdf", bbox_inches='tight')
plt.colorbar()
plt.gca().invert_yaxis()
```



Here we used a high number of principal components for PCA, and the companion is far away from the star, such that the

correlated speckle noise is efficiently subtracted from each reduced channel. Consequently, not much correlation can be seen between channels. Each spectral channel is most correlated to itself, and at most to its 2 closest neighbouring channels. Note that zero is adopted for uncorrelated channels.

6.2 2.2 Concatenating spectral correlation matrices

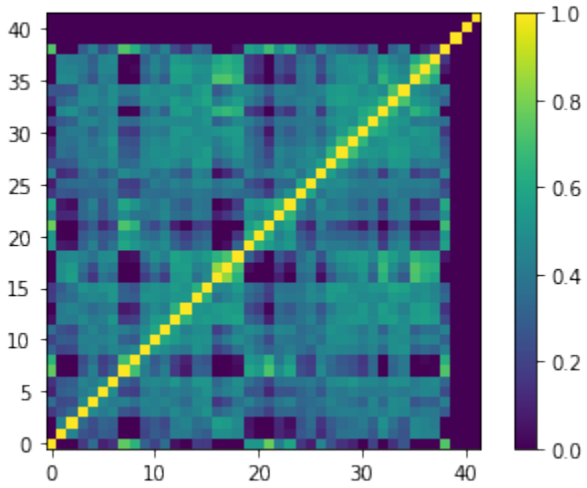
Since our full spectrum of CrA-9b/B also contains photometric points measured by SPHERE/IRDIS (2) and NACO (1), we need to pad the IFS spectral correlation matrix with 3 additional rows and columns of zeroes + ones on the diagonal, in order to get the final spectral correlation matrix.

This can easily be done with the `combine_spec_corrs` routine:

[16]:
```
from special.spec_corr import combine_spec_corrs

n_ird=2
n_naco=1
arr_list = [sp_corr]+[1]*n_ird+[1]*n_naco

final_sp_corr = combine_spec_corrs(arr_list)
```

Let's visualize the final matrix:

[17]:
```
plt.imshow(final_sp_corr)
plt.colorbar()
plt.gca().invert_yaxis()
```



Uncomment the lines in the following box if you wish to save the final spectral correlation matrix in fits format.

[18]:
```
#from special.fits import write_fits
#write_fits(datpath+"CrA-9_sp_corr_matrix.fits", final_sp_corr)
```

Go to the top

CHAPTER

SEVEN

3. PRELIMINARY SPECTRAL ANALYSIS

A number of utilities are implemented in the `utils_spec` and `spec_indices` module, which allow for a preliminary qualitative analysis of the spectrum of the companion. However, to avoid biased conclusions due to extinction in this preliminary analysis, let's first deredden the observed spectrum using the value quoted in the literature for CrA-9 ($A_V = 1.6$ mag; Dunham et al. 2015).

7.1 3.1 Dereddening

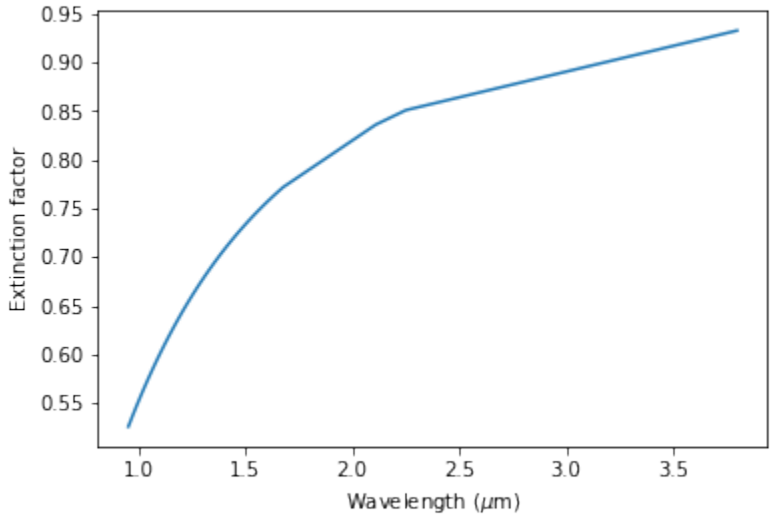
Provided a given A_V extinction value, the `extinction` routine can estimate the corresponding extinction factor at different wavelengths following the Cardelli et al. (1989) law, and for a desired R_V value. Let's consider the standard ISM value of $R_V = 3.1$ mag (default if not provided).

[19]:
```
from special.utils_spec import extinction
AV_est = 1.6 #mag
A_lambda = extinction(lbda, AV_est, RV=3.1)
extinc_fac = np.power(10,-A_lambda/2.5)
```

Let's plot the extinction factor as a function of wavelength:

[20]:
```
plt.plot(lbda, extinc_fac)
plt.xlabel(r"Wavelength ($\mu$m)")
plt.ylabel(r"Extinction factor")
```

[20]:
```
Text(0, 0.5, 'Extinction factor')
```

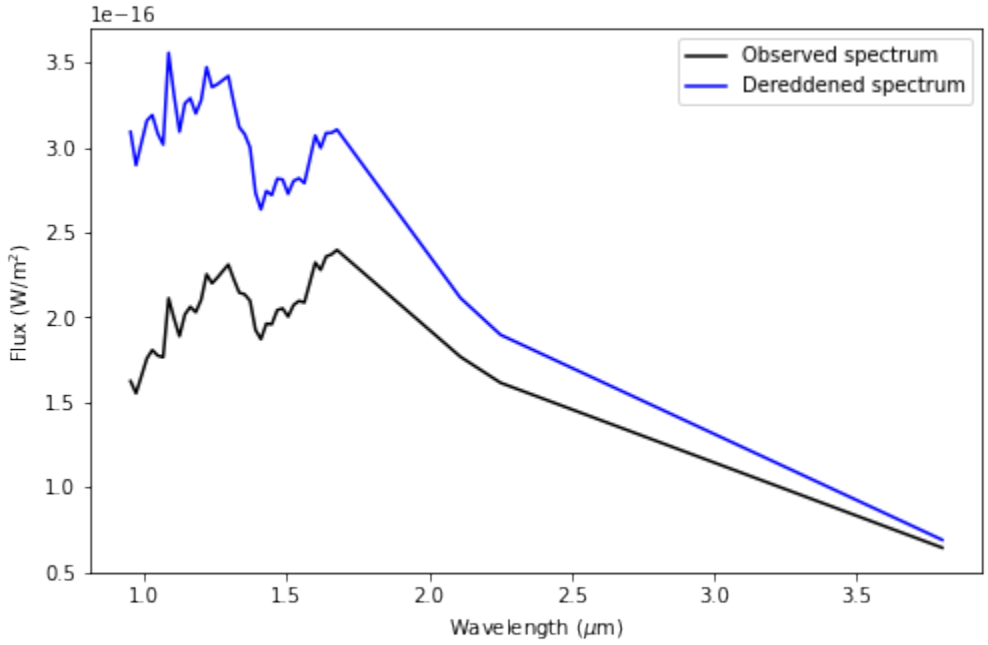


We see that as expected the effect of extinction is stronger at shorter NIR wavelengths: at 1.0 μm roughly only half of the flux is transmitted, while at L' band (3.8 μm) $\sim$ 93% of the flux is transmitted.

Let's now deredden the spectrum accordingly and plot it:

[21]:
```
spec_dered = spec/extinc_fac
spec_dered_err = spec_err/extinc_fac

fig = plt.figure(figsize=figsize)
plt.plot(lbda, lbda*spec, 'k', label="Observed spectrum")
plt.plot(lbda, lbda*spec_dered, 'b', label="Dereddened spectrum")
plt.xlabel(r"Wavelength ($\mu$m)")
plt.ylabel(r"Flux (W/m$^2$)")
plt.legend()
```

[21]:
```
<matplotlib.legend.Legend at 0x7fd0988406d0>
```



7.2 3.2 Spectral indices

(Sub)stellar objects of M, L and T types have been extensively studied. Some empirical spectral indices have been proposed to estimate either their spectral type or their gravity type from the measured flux in two parts of their spectrum. Some of these spectral indices have been included in `special` (additional inputs are welcome). An example is provided below with the H_2O and Na indices which are relevant for a YJH spectrum.

It is important to note that different spectral indices are valid in different spectral subtype ranges! The interested reader is referred to Allers et al. (2007) and references therein.

[22]:
```
from special.spec_indices import spectral_idx

H2O_idx = spectral_idx(lbda, spec_dered, 'H2O-1.5', spec_dered_err, verbose=True)
Na_idx = spectral_idx(lbda, spec_dered, 'Na-1.1', spec_dered_err, verbose=True)

print("**********************")
print("H2O index: {:.3f}+-{:.3f}".format(H2O_idx[0],H2O_idx[1]))
print("Na index: {:.3f}+-{:.3f}".format(Na_idx[0],Na_idx[1]))
print("**********************")
```

```
*WARNING*: input spectrum has VERY low spectral resolution.Closest (non-overlapping)␣
↪wavelengths to 1.150, 1.160, 1.134 and 1.144 mu correspond to 1.163, 1.163, 1.124 and␣
↪1.144 mu resp.Consider results with EXTRA CAUTION.
**********************
H2O index: 0.975+-0.007
Na index: 1.011+-0.004
**********************
```

The H_2O index can then be converted to an estimated spectral type:

[23]:
```
from special.spec_indices import sp_idx_to_spt, digit_to_spt

spt, spt_err = sp_idx_to_spt(H2O_idx[0], 'H2O-1.5', H2O_idx[1])
spt_str = digit_to_spt(spt, convention='Allers07')
print("SpT: {}+-{:.1f}".format(spt_str, spt_err))
```

```
SpT: M5.1+-0.8
```

The Na index on the other hand can be used to provide a qualitative estimate on the gravity (hence the youth) of an object:

[24]:
```
from special.spec_indices import sp_idx_to_gravity

sp_idx_to_gravity(Na_idx, name='Na-1.1')
```

```
The object's gravity is consistent with being very young (Na index less than 1.4; Allers␣
↪et al. 2007))
```

Go to the top

CHAPTER

EIGHT

4. MCMC SAMPLER EXAMPLES

Now let's fit different models to the observed spectrum of CrA-9 b/B.

In that purpose, let's first define some parameters that will be common to all MCMC sampler runs on this measured spectrum.

8.1 4.1. Parameters

Let's first define the distance to the system, which will be used to scale the models (along with the companion radius):

[25]:
```
d_st = 153. #pc based on Gaia
```

The internal workings of `special` allow for the convolution with the spectral PSF of an IFS, or the integration over a filter transmission curve (in the case of photometric points). For these options to be considered, let's provide the average resolving power of the IFS in YJH mode (34.5), and the name of the filenames where the filter transmission curves are encoded.

[26]:
```
instru_res = [34.5, 'SPHERE_IRDIS_D_K1.dat', 'SPHERE_IRDIS_D_K2.dat', 'CONICA_L_prime.dat
↪']
```

Now let's assign an index to each instrument and build a list that will tell the algorithm which `instru_res` values are to be assigned to which datapoints.

[27]:
```
n_ifs=39
instru_idx = np.array(n_ifs*[1]+[2]+[3]+[4])
```

In addition, we define a snippet function that can read the filter transmissions mentioned in the list. For convenience (and to allow for multiprocessing), it is defined in `utils.py`, and commented below FYI.

[28]:
```
from utils import filter_reader

# import pandas as pd
# def filter_reader(filename):
#     # snippet to read filter files and return a tuple: lbda (mu), transmission
#     rel_path = '../static/filters/'
#     filter_path = str((par_path / rel_path).resolve())+'/'
#     filt_table = pd.read_csv(filter_path+filename, sep=' ', header=0,
#                              skipinitialspace=True)
#     keys = filt_table.keys()
#     lbda_filt = np.array(filt_table[keys[0]])
#     if lbda_filt.dtype == 'O':
```

(continues on next page)

(continued from previous page)

```
#         filt_table = pd.read_csv(filter_path+filename, sep='\t', header=0,
#                                  skipinitialspace=True)
#         keys = filt_table.keys()
#         lbda_filt = np.array(filt_table[keys[0]])
#     elif lbda_filt.dtype == 'int32' or lbda_filt.dtype == 'int64':
#         lbda_filt = np.array(filt_table[keys[0]], dtype='float64')
#     if '(AA)' in keys[0]:
#         lbda_filt /=10000
#     elif '(mu)' in keys[0]:
#         pass
#     elif '(nm)' in keys[0]:
#         lbda_filt /=1000
#     else:
#         raise ValueError('Wavelength unit not recognised in filter file')
#     trans = np.array(filt_table[keys[1]])
#     return lbda_filt, trans def filter_reader(filename):
#     # snippet to read filter files and return a tuple: lbda (mu), transmission
#     rel_path = '../static/filters/'
#     filter_path = str((par_path / rel_path).resolve())+'/'
#     filt_table = pd.read_csv(filter_path+filename, sep=' ', header=0,
#                              skipinitialspace=True)
#     keys = filt_table.keys()
#     lbda_filt = np.array(filt_table[keys[0]])
#     if lbda_filt.dtype == 'O':
#         filt_table = pd.read_csv(filter_path+filename, sep='\t', header=0,
#                                  skipinitialspace=True)
#         keys = filt_table.keys()
#         lbda_filt = np.array(filt_table[keys[0]])
#     elif lbda_filt.dtype == 'int32' or lbda_filt.dtype == 'int64':
#         lbda_filt = np.array(filt_table[keys[0]], dtype='float64')
#     if '(AA)' in keys[0]:
#         lbda_filt /=10000
#     elif '(mu)' in keys[0]:
#         pass
#     elif '(nm)' in keys[0]:
#         lbda_filt /=1000
#     else:
#         raise ValueError('Wavelength unit not recognised in filter file')
#     trans = np.array(filt_table[keys[1]])
#     return lbda_filt, trans
```

Let's assemble all parameters related to the instrument(s) and filter(s) into a dictionary, including as well the spectral correlation calculated in Sec. 2:

[29]:
```
instru_params = {'instru_res':instru_res,
                 'instru_idx':instru_idx,
                 'instru_corr': final_sp_corr,
                 'filter_reader': filter_reader}
```

`special` was implemented with directly imaged substellar objects in mind. Radii and masses therefore implicitly assume Jovian units. If you prefer Solar radii for your outputs, set `star` to True in the following box:

[30]:
```
from astropy import constants as c

star = False # whether use Solar units
if star:
    conv_RS = c.R_jup/c.R_sun
    conv_RS
```

Let's set the MCMC parameters identically for all runs shown in the next subsections (see more details in `emcee` documentation):

[31]:
```
a=2.0
nwalkers=100
niteration_min=100
niteration_limit=500 # recommended>> 1000
nproc = 2            # number of CPUs

mcmc_params = {'a':a,
               'nwalkers':nwalkers,
               'niteration_min':niteration_min,
               'niteration_limit':niteration_limit,
               'nproc':nproc}
```

Similarly, we adopt the autocorrelation time criterion for convergence for all MCMC runs (see more details here):

[32]:
```
conv_test='ac'
ac_c=50
ac_count_thr=1

conv_params = {'conv_test': conv_test,
               'ac_c': ac_c,
               'ac_count_thr':ac_count_thr}
```

Note that for the purpose of this tutorial, we set a low number of walkers and a low maximum number of iterations. It is recommended to set it large enough (e.g. >>1000, although it depends on the considered number of parameters and model) such that the convergence criterion will break the chain once it is met.

8.2 4.2. Blackbody model

This model does not involve any grid:

[33]:
```
grid_list=None # triggers a blackbody fit
```

The 2 parameters are the temperature and radius (of a sphere). It is important to label the blackbody temperature and radius 'Tbb1' and 'Rbb1', respectively. This is because `special` allows for the addition of an unlimited number of blackbody components. E.g. for n components, set the labels from 'Tbb1' to 'Tbbn' and from 'Rbb1' to 'Rbbn'.

[34]:
```
labels = ('Tbb1', 'Rbb1')
units = ('K', r'$R_J$')

npar = len(labels)
```

Set the initial guesses and bounds accordingly. Note the requirement for a Jovian input radius.

[35]:
```
ini_guess = (3200., 1.3) #K and R_Jup
bounds = {'Tbb1':(2000,4000),
          'Rbb1':(0.1,5)}
```

Again, let's collate all model-related parameters into a dictionary:

[36]:
```
model_params={'grid_param_list':grid_list,
              'labels':labels,
              'initial_state':ini_guess,
              'bounds':bounds}
```

The flux units of input spectra are assumed to follow the SI convention by default (W $m^{-2}\mu m^{-1}$). Note that `special` can also handle 'cgs' and 'jy'. The relevant conversions will be made internally if different units are provided for observed and model spectra.

[37]:
```
units_obs = 'si'
```

Now let's run the MCMC sampler (**Warning: the next cell may take a few minutes to complete**):

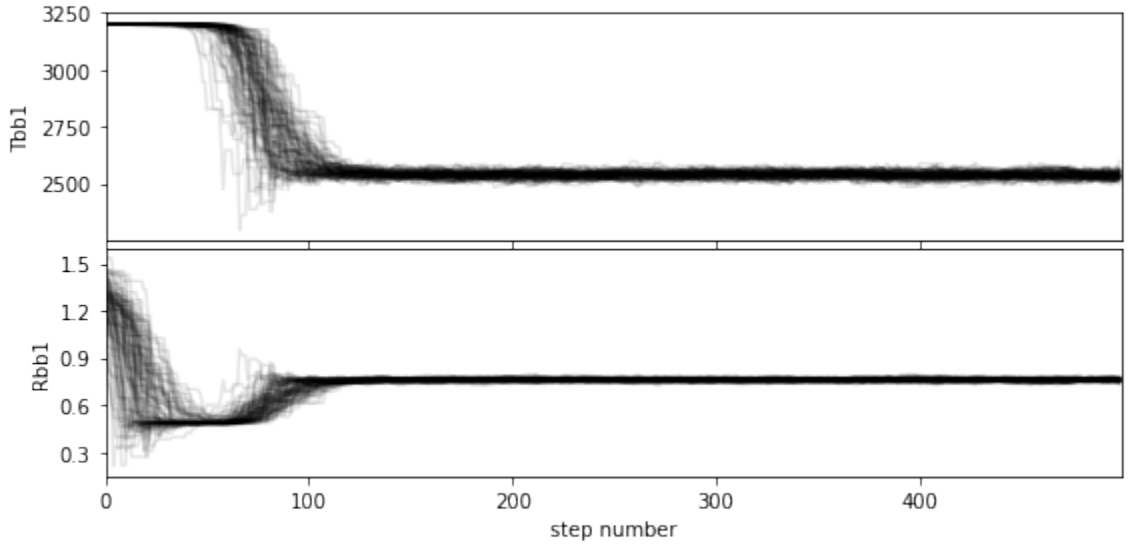
[38]:
```
from special.mcmc_sampling import mcmc_spec_sampling

res = mcmc_spec_sampling(lbda, spec, spec_err, d_st, dlbda_obs=dlbda, units_obs=units_
↪obs,
                         **instru_params, **mcmc_params, **conv_params, **model_params)
```

Let's convert the sampled radii into Solar radii (if requested):

[39]:
```
final_chain, ln_proba = res
if star:
    idx_R = labels.index('Rbb1')
    final_chain[:,:,idx_R] = final_chain[:,:,idx_R].copy()*conv_RS
    # change units of radius
    units = list(units)
    units[-1] = r'$R_{\odot}$'
    units = tuple(units)
```

Let's inspect the walk plot:

[40]:
```
from special.mcmc_sampling import show_walk_plot

show_walk_plot(final_chain, labels)#, save=False,# output_dir='../datasets/',
```



If you wish to save the walk plot in a pdf, set `save=True` and provide a value for the output directory (`output_dir`).

Based on the walk plot, let's define a conservative burnin factor of 0.3 and apply it to the chain:

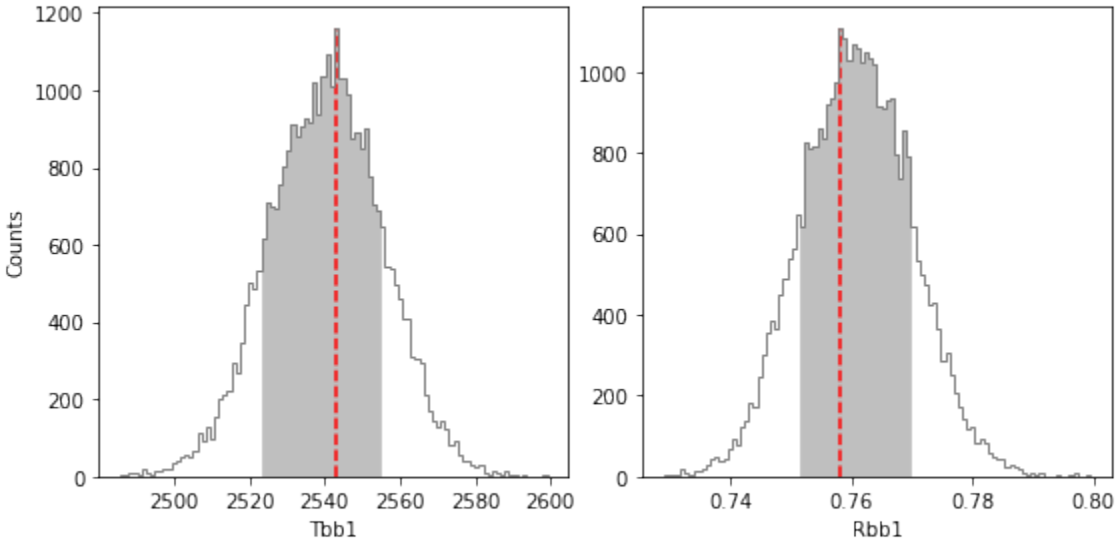
[41]:
```
burnin=0.3
```

[42]:
```
cutoff = int(final_chain.shape[1]//(1/burnin))
ngood_steps = final_chain.shape[1]-cutoff
samples_flat_BB = final_chain[:, cutoff:, :].reshape((-1,npar))
#write_fits(output_dir+"isamples_flat.fits",isamples_flat)
```

Let's compute the 68.27% confidence intervals on the posterior distribution, after burnin:

[43]:
```
from special.mcmc_sampling import confidence

bins = 100
vals, err = confidence(samples_flat_BB, labels, cfd=68.27, bins=bins,
                           gaussian_fit=False, weights=None,
                           verbose=True, save=False, bounds=bounds,
                           priors=None)
```

```
percentage for Tbb1: 68.70571428571428%
percentage for Rbb1: 68.51428571428573%
******* Results for Tbb1 *****


Confidence intervals:
Tbb1: 2542.9954443350794 [-19.820137461919785,11.892082477151689]
******* Results for Rbb1 *****


Confidence intervals:
Rbb1: 0.7582530539355073 [-0.0066963793890129075,0.011630553675653799]
```



If you wish to save the results in a text file, set `save=True` and provide a value for the output directory (`output_dir`).

Let's define the most likely parameters along with their uncertainties in a numpy array:

[44]:
```
post_params = np.zeros([npar,3])
for i in range(npar):
    post_params[i,0] = vals[labels[i]]
    post_params[i,1] = err[labels[i]][0]
    post_params[i,2] = err[labels[i]][1]
```

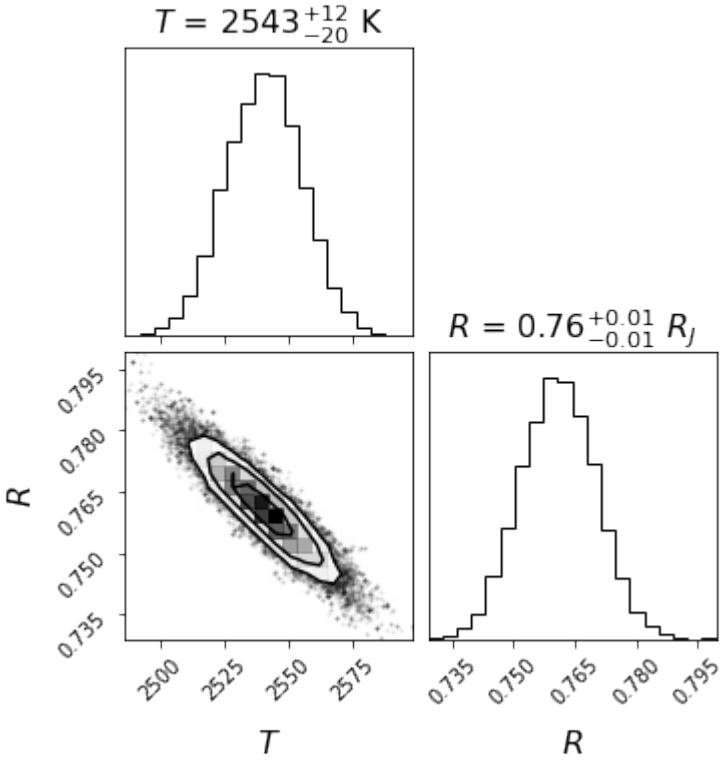
Now, let's have a look at the corner plot. A couple of parameters can be fine tuned for aesthetics, e.g.:

- number of significant digits for each parameter (`ndig`),
- labels to be used in the plot for each parameter (`labels_plot` can be different to the string used in `labels` but `labels` is used if not provided),
- font attributes for plot title and label,
- number of bins to consider for the corner plot histograms.

[45]:
```
ndig = (0,2) # should have same length as labels
labels_plot = (r'$T$', r'$R$')
title_kwargs={"fontsize": 16}
label_kwargs={"fontsize": 16}
corner_bins = 20
```

If you wish to save the corner plot in pdf format, set `save=True` and provide a value to `plot_name`.

[46]:
```
from special.mcmc_sampling import show_corner_plot

#note: provide the chain before burnin, since burnin is done internally here:
show_corner_plot(final_chain, burnin=burnin, mcmc_res=post_params, units=units,
                 ndig=ndig, labels_plot=labels_plot, labels=labels,
                 #save=True, plot_name='corner_plot.pdf',
                 title_kwargs=title_kwargs, label_kwargs=label_kwargs, bins=corner_bins)
```



As expected, the 2 parameters are highly dependent on each other. With only 2 parameters, we also note that the estimated uncertainties on each parameter are very small. As we will see, with a more realistic model including more parameters, the uncertainties on each parameter get wider.

Let's save the likelihood and parameter values of the most likely model for later (*Sec. 5.2*):

[47]:
```
# favoured model has highest likelihood
max_prob_BB = np.nanmax(ln_proba)
idx_max = np.unravel_index(np.nanargmax(ln_proba),shape=ln_proba.shape)
bf_params_BB = final_chain[idx_max]
bf_params_BB
```

[47]:
```
array([2.53994358e+03, 7.61007960e-01])
```

8.3 4.3. BT-SETTL model

Let's now consider a grid of BT-SETTL models. In addition, let's also consider extinction as a free parameter, since for young objects it is possible that different amounts of dust surround the companion and the central star.

In the case of BT-SETTL models, there are 2 free parameters in the grid: effective temperature and surface gravity. Let's define the grid of parameter values covered by the models in our possession (in the `models/btsettl15/` directory).

[48]:
```
teff_list = np.linspace(2000,4000,num=21).tolist()
logg_list = np.linspace(2.5,5.5,num=7).tolist()
grid_list = [teff_list, logg_list]
```

Let's now define a snippet function that can read the input files of the grid, and provide as output a tuple of 2 vectors

(1d arrays) for the wavelength and flux (in SI units, at the surface of the object). For convenience (and to allow for multiprocessing), it is defined in `utils.py`, and commented below FYI.

[49]:
```
from utils import bts_reader
# def bts_reader(params, ground=True):
#     # snippet to read BT-SETTL fits files and return a tuple: lbda (mu), spec (SI)
#     rel_path = '../static/btsettl15_models/'
#     mod_path = str((par_path / rel_path).resolve())+'/'

#     mod_units = 'cgs' # careful: input in cgs
#     filename = mod_path+'btsettl15_t{:.0f}_g{:.1f}0_z-0.00_SED.txt'.format(params[0],
↪params[1])
#     lbda = []
#     flux = []
#     f=open(filename,"r")
#     lines=f.readlines()
#     for i, x in enumerate(lines):
#         if i>0:
#             lbda.append(float(x.split('\t')[0]))
#             flux.append(float(x.split('\t')[1]))
#     f.close()

#     # Correct for atmospheric refraction
#     flux = np.array(flux)
#     lbda = np.array(lbda)
#     if ground:
#         # truncate at 100nm (no transmission)
#         flux = flux[np.where(lbda>0.2)]
#         lbda = lbda[np.where(lbda>0.2)]
#         # then correct for the shift in wavelength due to atm. refraction
#         nref = nrefrac(lbda*1e4)
#         lbda = lbda/(1+(nref*1e-6))

#     #conversion from ergs/s/cm2/um to W/m2/um
#     flux = convert_F_units(flux, lbda, in_unit=mod_units, out_unit='si')

#     return lbda, flux
```

Since the snippet function already performs the conversion to SI flux units (W $m^{-2}\mu m^{-1}$), we will set the model units to 'si'. Note that 'cgs' units or 'jy' are also accepted - the conversion to match the observed flux would then be made internally.

[50]:
```
units_mod = 'si'
```

Considering the extinction `A_V` and photometric radius `R` (in Jovian radii) as free parameters, we have a total of 4 free parameters, including 2 captured by the grid: the temperature and surface gravity.

[51]:
```
labels = ('Teff', 'logg', 'R', 'Av')
units = ('K', '', r'$R_J$', 'mag')

npar = len(labels)
```

Set the initial guesses and bounds accordingly. Note the requirement for a Jovian input radius.

[52]:
```
ini_guess = (3100., 3.5, 1.3, 1.6)
bounds = {'Teff':(2000,4000),
          'logg':(2.5,5.5),
          'R':(0.1,5),
          'Av':(0.,5)}
```

[53]:
```
model_params={'grid_param_list':grid_list,
              'model_reader':bts_reader,
              'labels':labels,
              'initial_state':ini_guess,
              'bounds':bounds,
              'units_mod':units_mod}
```

In order for the MCMC sampler to run faster, all models of the grid are first resampled (at the same sampling as the measured spectrum), such that only light spectra files are opened and interpolated during the sampling. To do so, set `resamp_before=True` and provide a path and name where the resampled grid will be saved:

[54]:
```
grid_name = 'bts_resamp_grid.fits'
rel_path = '../static/bts_output/'
output_dir = str((par_path / rel_path).resolve())+'/'
resamp_before=True
```

Important: if a file named `output_dir+grid_name` already exists it will be used, instead of creating a new resampled grid (this is to save time when a given MCMC sampling is performed multiple times with different options). Make sure to change either `output_dir` or `grid_name` when considering different models or a different range of parameter values in the grid (i.e. differente `grid_list`).

We also set a filename where the MCMC results will be written as a pickle, such that they can be accessed without having to run again the MCMC.

[55]:
```
output_file = 'MCMC_results' # also written in output_dir

output_params = {'resamp_before':resamp_before,
                 'grid_name':grid_name,
                 'output_dir': output_dir,
                 'output_file': output_file}
```

Now let's run the MCMC sampler (**Warning: the next cell may take a few minutes to complete**):

[56]:
```
from special.mcmc_sampling import mcmc_spec_sampling

res = mcmc_spec_sampling(lbda, spec, spec_err, d_st, dlbda_obs=dlbda,
                         **instru_params, **mcmc_params, **conv_params,
                         **model_params, **output_params)
```

```
Fits HDU-0 data successfully loaded. Data shape: (21, 7, 42, 2)
```

[57]:
```
final_chain, ln_proba = res
if star:
    idx_R = labels.index('R')
    final_chain[:,:,idx_R] = final_chain[:,:,idx_R].copy()*conv_RS
    # change units of radius
    units = list(units)
```

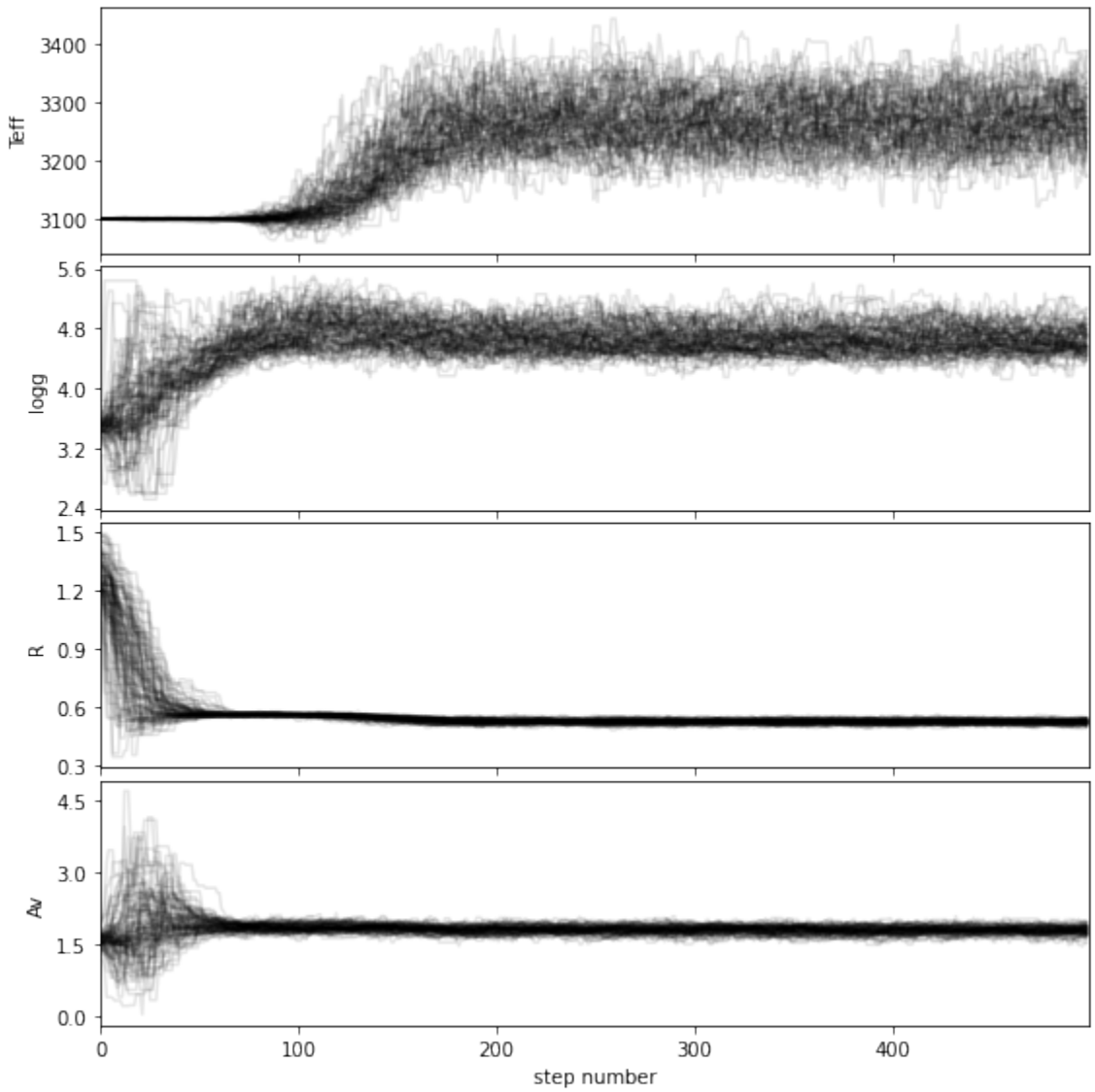
(continues on next page)

(continued from previous page)

```
    units[-1] = r'$R_{\odot}$'
    units = tuple(units)
```

Let's inspect the walk plot:

[58]:
```
from special.mcmc_sampling import show_walk_plot

show_walk_plot(final_chain, labels)#, save=False,# output_dir='../datasets/',
```



If you wish to save the walk plot in a pdf, set `save=True` and provide a value for the output directory (`output_dir`).

Based on the walk plot, let's define a conservative burnin factor of 0.3 and apply it to the chain:

[59]:
```
burnin=0.3
```

[60]:
```
cutoff = int(final_chain.shape[1]//(1/burnin))
```

(continues on next page)

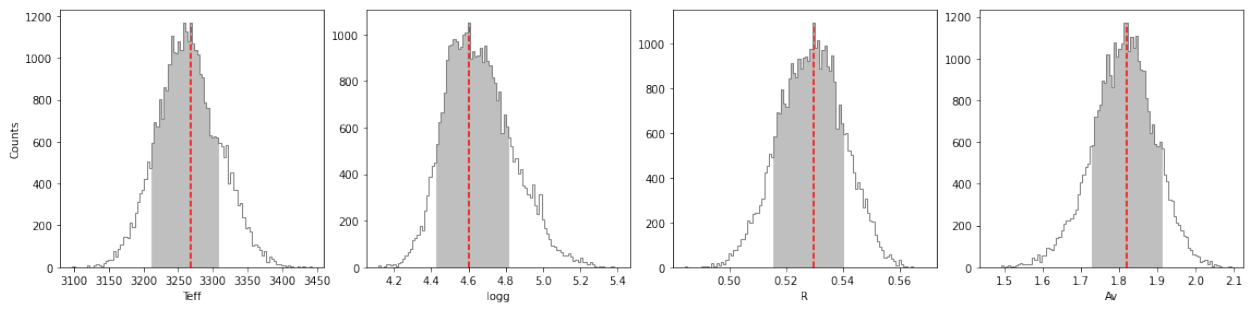
(continued from previous page)

```
ngood_steps = final_chain.shape[1]-cutoff
samples_flat_BTS = final_chain[:, cutoff:, :].reshape((-1,npar))
#write_fits(output_dir+"isamples_flat.fits",isamples_flat)
```

Let's compute the 68.27% confidence intervals on the posterior distribution, after burnin:

[61]:
```
from special.mcmc_sampling import confidence

bins = 100
vals, err = confidence(samples_flat_BTS, labels, cfd=68.27, bins=bins,
                           gaussian_fit=False, weights=None,
                           verbose=True, save=False, bounds=bounds,
                           priors=None)
```

```
percentage for Teff: 69.57714285714285%
percentage for logg: 69.28%
percentage for R: 69.23142857142857%
percentage for Av: 68.71714285714286%
******* Results for Teff *****


Confidence intervals:
Teff: 3268.537193631018 [-56.98597566100216,39.717498187971614]
******* Results for logg *****


Confidence intervals:
logg: 4.600436767117264 [-0.17341985736505894,0.21195760344618364]
******* Results for R *****


Confidence intervals:
R: 0.5296112793509183 [-0.013945448920862469,0.010564734030956 48]
******* Results for Av *****


Confidence intervals:
Av: 1.8199430173064597 [-0.09084551385115014,0.09084551385115036]
```



If you wish to save the results in a text file, set `save=True` and provide a value for the output directory (`output_dir`).

Let's define the most likely parameters along with their uncertainties in a numpy array:

[62]:
```
post_params = np.zeros([npar,3])
for i in range(npar):
    post_params[i,0] = vals[labels[i]]
    post_params[i,1] = err[labels[i]][0]
    post_params[i,2] = err[labels[i]][1]
```

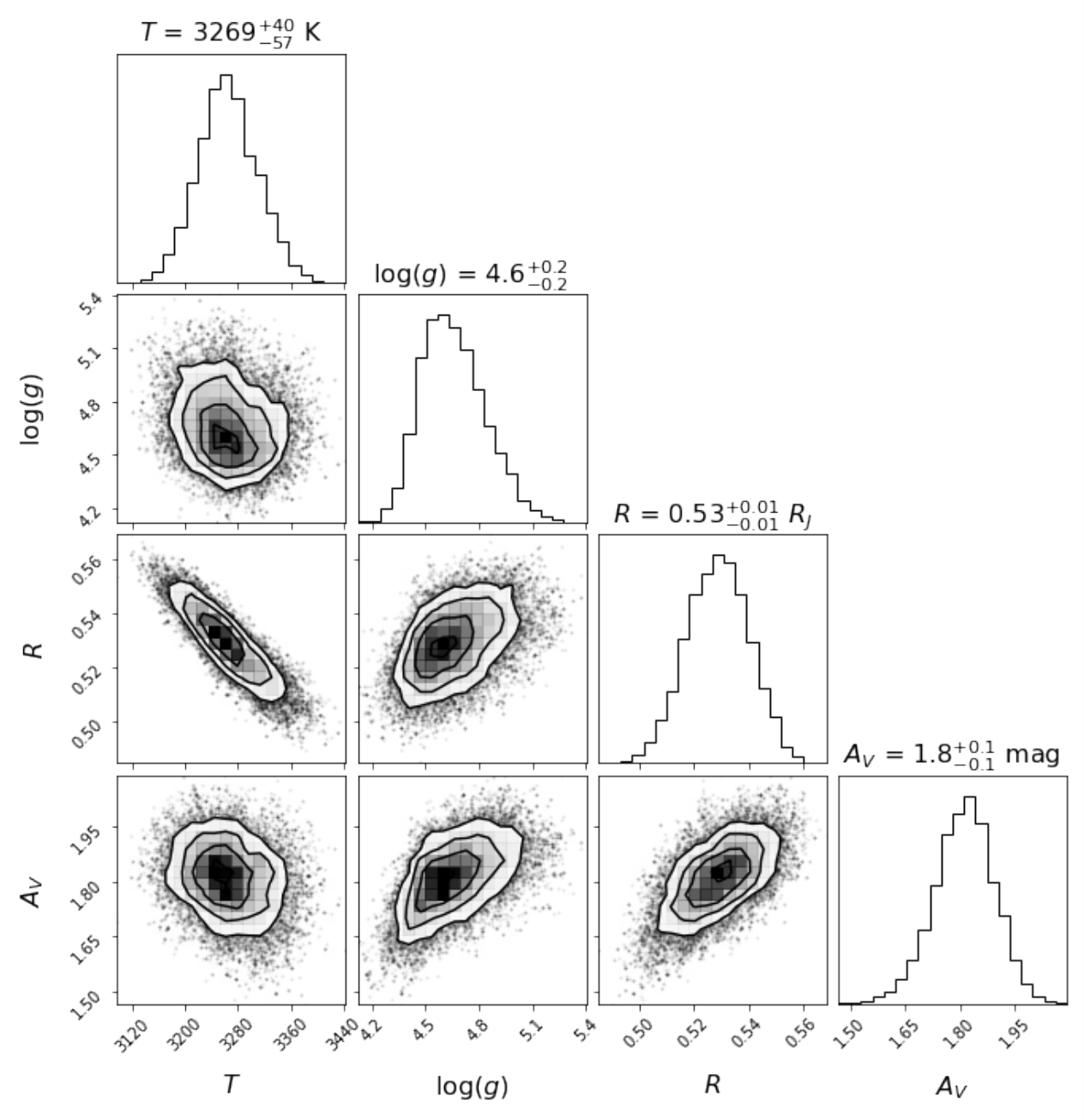
Now, let's have a look at the corner plot. A couple of parameters can be fine tuned for aesthetics, e.g.:

- number of significant digits for each parameter (`ndig`),
- labels to be used in the plot for each parameter (`labels_plot` can be different to the string used in `labels` but `labels` is used if not provided),
- font attributes for plot title and label,
- number of bins to consider for the corner plot histograms.

[63]:
```
ndig = (0,1,2,1) # should have same length as labels
labels_plot = (r'$T$', r'log($g$)', r'$R$',  r'$A_V$')
title_kwargs={"fontsize": 16}
label_kwargs={"fontsize": 16}
corner_bins = 20
```

If you wish to save the corner plot in pdf format, set `save=True` and provide a value to `plot_name`.

[64]:
```
from special.mcmc_sampling import show_corner_plot

#note: provide the chain before burnin, since burnin is done internally here:
show_corner_plot(final_chain, burnin=burnin, mcmc_res=post_params,
                 units=units, ndig=ndig, labels_plot=labels_plot,
                 labels=labels, #save=True, plot_name='corner_plot.pdf',
                 title_kwargs=title_kwargs, label_kwargs=label_kwargs, bins=corner_bins)
```



Let's save the likelihood and parameter values of the most likely model for later (*Sec. 5.2*):

[65]:
```
# favoured model has highest likelihood
max_prob_BTS = np.nanmax(ln_proba)
idx_max = np.unravel_index(np.nanargmax(ln_proba),shape=ln_proba.shape)
bf_params_BTS = final_chain[idx_max]
bf_params_BTS
```

[65]:
```
array([3.27776316e+03, 4.51097245e+00, 5.22860539e-01, 1.77473043e+00])
```

The results above are puzzling: a ~3000 K effective temperature combined with a ~0.5 R_J photometric radius. See Christiaens et al. (2021) for a discussion on the different hypotheses put forward to reconcile these results.

8.4 4.4. BT-SETTL + BB model

Although this is not necessarily a good model for the NIR spectrum of CrA-9 B/b, let's illustrate how to set up a photospheric + extra blackbody component model to further showcase the possibilities available in `special`.

Let's set Gaussian priors on the temperatures of each Blackbody component, to replicate the kinds of constraints that could potentially be obtained from past works.

Note that a single extra BB component is considered here, but `special` does not limit the number of components you wish to incorporate in the model (just add the desired number of parameters 'Tbb*n*' and 'Rbb*n*', where n>0).

This time, we have a total of 8 free parameters, with again only 2 captured by the BT-SETTL grid:

[66]:
```
labels = ('Teff', 'logg', 'R', 'Av', 'Tbb1', 'Rbb1')
units = ('K', '', r'$R_J$', 'mag', 'K', r'$R_J$')
npar = len(labels)

teff_list = np.linspace(2000,4000,num=21).tolist()
logg_list = np.linspace(2.5,5.5,num=7).tolist()
grid_list = [teff_list, logg_list]
```

Set the initial guesses and bounds accordingly. Note the requirement for a Jovian input radius.

[67]:
```
ini_guess = (3100., 3.5, 1.3, 1.6, 1100, 1)
bounds = {'Teff':(2000,4000),
          'logg':(2.5,5.5),
          'R':(0.1,5),
          'Av':(0.,5),
          'Tbb1':(900,2000),
          'Rbb1':(0.1,5)}
```

Let's set Gaussian priors for $\log(g)$ and T_{bb1} in a dictionary. By default, uniform priors are considered for parameters not mentioned in this dictionary. Be sure to use the same dictionary labels as for parameter `labels`.

[68]:
```
priors = {'logg':(3.5,0.2),
          'Tbb1':(1200,200)}
```

When including a blackbody component, you can ensure that the sampled solutions are physical (e.g. the BB temperatures must systematically be lower temperature than the photosphere effective temperature - assuming it comes from surrounding heated dust):

[69]:
```
physical=True
```

It is also possible to tell the model whether to apply the extinction before or after adding the blackbody component, depending on where the dust absorbing the companion signal is to be assumed with respect to the emitting dust. By default, we assume the latter (e.g. interstellar dust).

[70]:
```
AV_bef_bb = False
```

Let's compile all model-related parameters defined above into a dictionary:

[71]:
```
model_params={'grid_param_list':grid_list,
              'model_reader':bts_reader,
              'labels':labels,
              'initial_state':ini_guess,
```

(continues on next page)

(continued from previous page)

```
               'bounds':bounds,
               'priors':priors,
               'units_mod':units_mod,
               'physical':physical,
               'AV_bef_bb':AV_bef_bb}
```

In order for the MCMC sampler to run faster, all models of the grid are first resampled (at the same sampling as the measured spectrum), such that only light spectra files are opened and interpolated during the sampling. To do so, set `resamp_before=True` and provide a path and name where the resampled grid will be saved:

[72]:
```
grid_name = 'bts_bb_resamp_grid.fits'
rel_path = '../static/bts_output/'
output_dir = str((par_path / rel_path).resolve())+'/'
resamp_before=True
```

Important: if a file named `output_dir+grid_name` already exists it will be used, instead of creating a new resmpled grid (this is to save time when a given MCMC sampling is performed multiple times with different options). Make sure to change either `output_dir` or `grid_name` when considering different models or a different range of parameter values in the grid (i.e. different `grid_list`).

We also set a filename where the MCMC results will be written as a pickle, such that they can be accessed without having to run again the MCMC.

[73]:
```
output_file = 'MCMC_results_bb' # also written in output_dir

output_params = {'resamp_before':resamp_before,
                 'grid_name':grid_name,
                 'output_dir': output_dir,
                 'output_file': output_file}
```

Finally, given the larger number of parameters, let's increase the number of iterations and walkers:

[74]:
```
a=2.0
nwalkers=200
niteration_min=100
niteration_limit=1000 # recommended>> 1000
nproc = 2             # number of CPUs

mcmc_params = {'a':a,
               'nwalkers':nwalkers,
               'niteration_min':niteration_min,
               'niteration_limit':niteration_limit,
               'nproc':nproc}
```

Now let's run the MCMC sampler (**Warning: the next cell may take a few minutes to complete**):

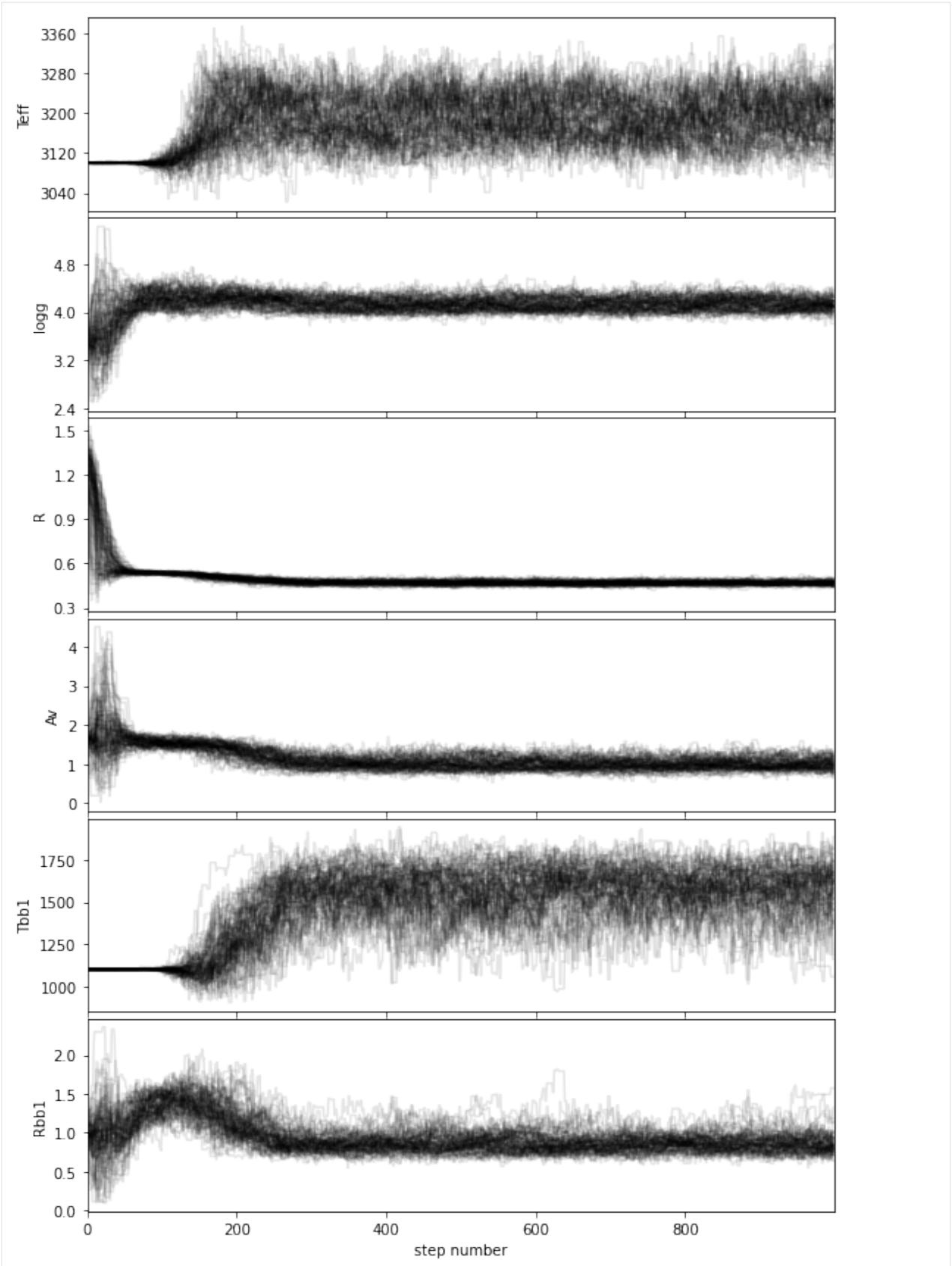
[75]:
```
from special.mcmc_sampling import mcmc_spec_sampling

res = mcmc_spec_sampling(lbda, spec, spec_err, d_st, dlbda_obs=dlbda, units_obs=units_
↪obs,
                         **instru_params, **mcmc_params, **conv_params, **model_params,
                         **output_params)
```

```
Fits HDU-0 data successfully loaded. Data shape: (21, 7, 42, 2)
```

[76]:
```
final_chain, ln_proba = res
if star:
    idx_R = labels.index('R')
    final_chain[:,:,idx_R] = final_chain[:,:,idx_R].copy()*conv_RS
    # change units of radius
    units = list(units)
    units[-1] = r'$R_{\odot}$'
    units = tuple(units)
```

Let's inspect the walk plot:

[77]:
```
from special.mcmc_sampling import show_walk_plot

show_walk_plot(final_chain, labels)#, save=False,# output_dir='../datasets/',
```


Teff
3360
3280
3200
3120
3040
logg
4.8
4.0
3.2
2.4
R
1.5
1.2
0.9
0.6
0.3
Av
4
3
2
1
0
Tbb1
1750
1500
1250
1000
Rbb1
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
step number

If you wish to save the walk plot in a pdf, set `save=True` and provide a value for the output directory (`output_dir`).

Based on the walk plot, let's define a conservative burnin factor of 0.5 and apply it to the chain:

[78]:
```
burnin=0.5
```

[79]:
```
cutoff = int(final_chain.shape[1]//(1/burnin))
ngood_steps = final_chain.shape[1]-cutoff
samples_flat_BTS_BB = final_chain[:, cutoff:, :].reshape((-1,npar))
```

Let's compute the 68.27% confidence intervals on the posterior distribution, after burnin:

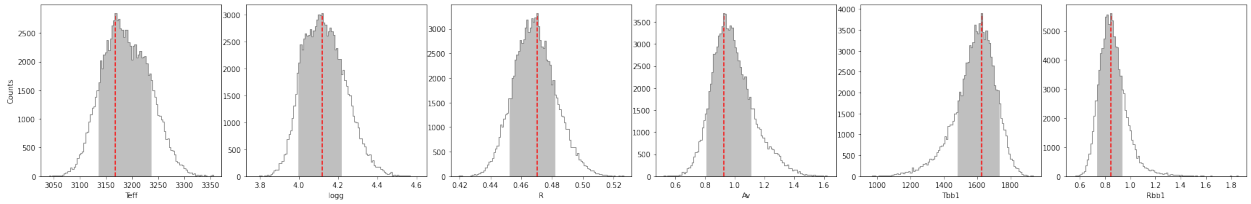
[80]:
```
from special.mcmc_sampling import confidence

bins = 100
vals, err = confidence(samples_flat_BTS_BB, labels, cfd=68.27, bins=bins,
                           gaussian_fit=False, weights=None,
                           verbose=True, save=False, bounds=bounds,
                           priors=None)
```

```
percentage for Teff: 69.11699999999999%
percentage for logg: 68.27300000000001%
percentage for R: 69.448%
percentage for Av: 70.02799999999996%
percentage for Tbb1: 70.16400000000002%
percentage for Rbb1: 68.334%
******* Results for Teff *****


Confidence intervals:
Teff: 3168.8061684879867 [-32.93135392899421,67.43086756889261]
******* Results for logg *****


Confidence intervals:
logg: 4.119528892365588 [-0.12154198984640097,0.09639537125749076]
******* Results for R *****


Confidence intervals:
R: 0.47029462276193956 [-0.017623047379509982,0.01121466651423364]
******* Results for Av *****


Confidence intervals:
Av: 0.9262609450612325 [-0.1161701801150623,0.18255314018081226]
******* Results for Tbb1 *****


Confidence intervals:
Tbb1: 1625.0735030189098 [-142.87325865375692,103.45994592168563]
******* Results for Rbb1 *****
```

(continues on next page)

(continued from previous page)

```
Confidence intervals:
Rbb1: 0.8454529974913282 [-0.11071662338551935,0.08466565317716201]
```



If you wish to save the results in a text file, set `save=True` and provide a value for the output directory (`output_dir`).

Let's define the most likely parameters along with their uncertainties in a numpy array:

[81]:
```
post_params = np.zeros([npar,3])
for i in range(npar):
    post_params[i,0] = vals[labels[i]]
    post_params[i,1] = err[labels[i]][0]
    post_params[i,2] = err[labels[i]][1]
```

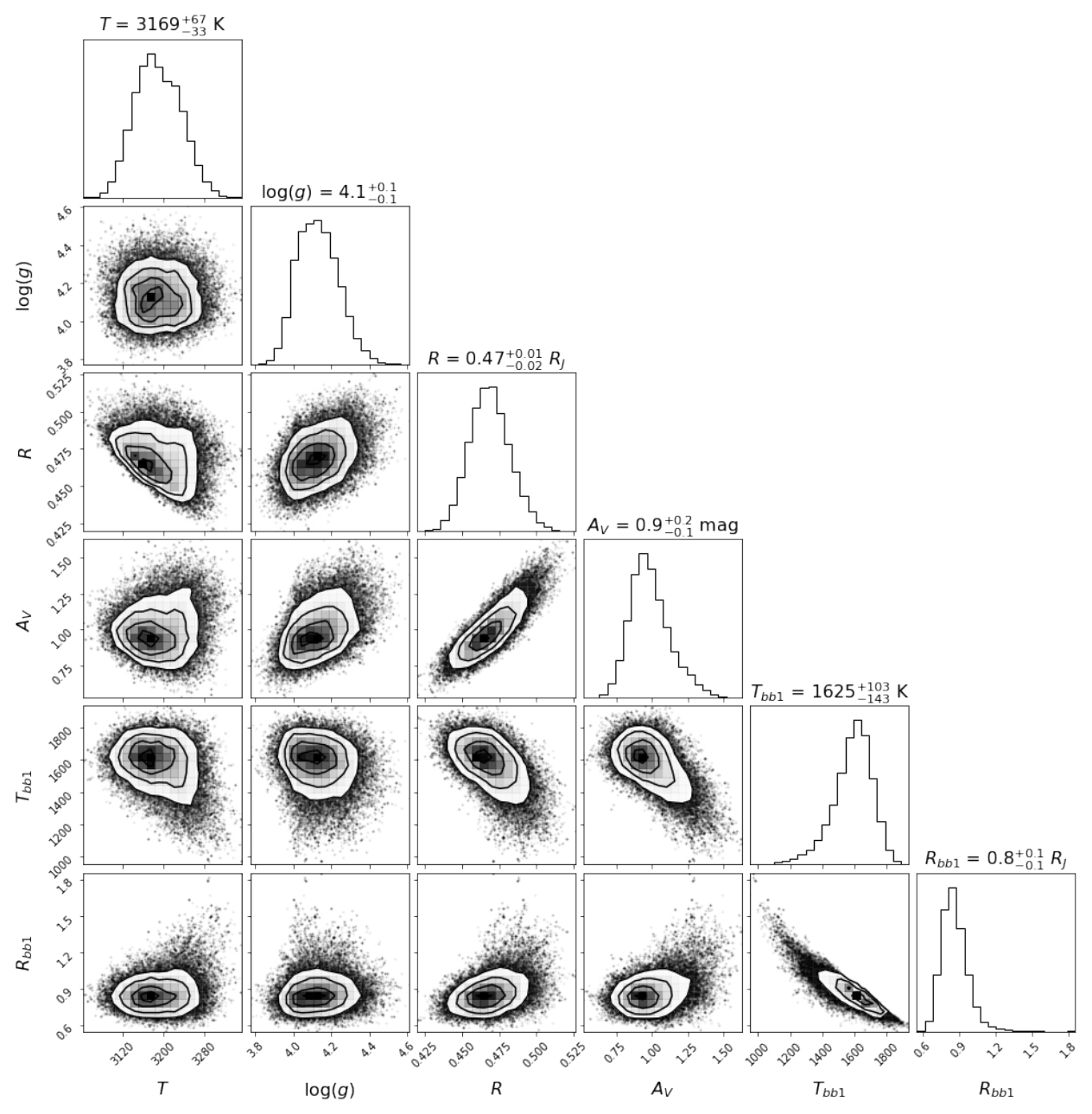
Now, let's have a look at the corner plot. A couple of parameters can be fine tuned for aesthetics, e.g.:

- number of significant digits for each parameter (`ndig`),
- labels to be used in the plot for each parameter (`labels_plot` can be different to the string used in `labels` but `labels` is used if not provided),
- font attributes for plot title and label,
- number of bins to consider for the corner plot histograms.

[82]:
```
ndig = (0,1,2,1,0,1) # should have same length as labels
labels_plot = (r'$T$', r'log($g$)', r'$R$',  r'$A_V$', r'$T_{bb1}$',r'$R_{bb1}$')
title_kwargs={"fontsize": 16}
label_kwargs={"fontsize": 16}
corner_bins = 20
```

If you wish to save the corner plot in pdf format, set `save=True` and provide a value to `plot_name`.

[83]:
```
from special.mcmc_sampling import show_corner_plot

#note: provide the chain before burnin, since burnin is done internally here:
show_corner_plot(final_chain, burnin=burnin, mcmc_res=post_params, units=units,
                 ndig=ndig, labels_plot=labels_plot,
                 labels=labels, #save=True, plot_name='corner_plot.pdf',
                 title_kwargs=title_kwargs, label_kwargs=label_kwargs, bins=corner_bins)
```



Let's save the likelihood and parameter values of the most likely model for later (*Sec. 5.2*):

[84]:
```
# favoured model has highest likelihood
max_prob_BTS_BB = np.nanmax(ln_proba)
idx_max = np.unravel_index(np.nanargmax(ln_proba),shape=ln_proba.shape)
bf_params_BTS_BB = final_chain[idx_max]
bf_params_BTS_BB
```

[84]:
```
array([3.15611462e+03, 4.00105754e+00, 4.59765800e-01, 8.45928708e-01,
       1.66216448e+03, 8.28073620e-01])
```

8.5 4.5. BT-SETTL model + BrG line model

In order to further showcase the possibilities available in `special`, let's now consider a BT-SETTL + Brackett Gamma line model for the spectrum. Let's also consider R_V as a free parameter, and finally consider Gaussian priors for $\log(g)$ and R_V: $\mu_{\log(g)} = 3.5$, $\sigma_{\log(g)} = 0.2$ and $\mu_{R_V} = 3.1$, $\sigma_{R_V} = 0.2$.

This time, we have a total of 6 free parameters, including 2 captured by the BT-SETTL grid:

[85]:
```
labels = ('Teff', 'logg', 'R', 'Av', 'Rv', 'BrG')
units = ('K', '', r'$R_J$', 'mag', '', '')
npar = len(labels)

teff_list = np.linspace(2000,4000,num=21).tolist()
logg_list = np.linspace(2.5,5.5,num=7).tolist()
grid_list = [teff_list, logg_list]
```

We first provide an estimate for the possible BrG flux:

[86]:
```
import astropy.constants as con
F_BrG_estimate = 20000 # W m-2 at the surface (based on the estimate for the primary)
RB_est = 5             # rough estimate for the radius of the companion in R_J
input_unit = 'LogL'    # select input units {'F','L','LogL'} for Flux (W/m2), Luminosity␣
↪(W) or Log Solar Luminosity.
```

We then convert it into log Solar luminosity by setting the `input_unit` to 'LogL'. Other formats are accepted, such as flux 'F' in SI units, or Solar luminosity 'L'.

[87]:
```
if input_unit == 'F':
    BrG_min = F_BrG_estimate*1e-5
    BrG_max = F_BrG_estimate*1e5
elif input_unit == 'LogL':
    # BrG flux is converted below into L_Sun
    L_BrG_estimate = np.log10(4*np.pi*np.power(con.R_jup.value*RB_est,2)*F_BrG_estimate/
↪con.L_sun.value)
    # set min and max of BrG list of initial grid of tested values
    BrG_min = L_BrG_estimate-5
    BrG_max = L_BrG_estimate+5
elif input_unit == 'L':
    BrG_min = F_BrG_estimate*1e-5*4*np.pi*np.power(con.R_jup.value*RB_est,2)
    BrG_max = F_BrG_estimate*1e5*4*np.pi*np.power(con.R_jup.value*RB_est,2)

print(BrG_min,BrG_max)
```

```
-9.076277167361013 0.9237228326389868
```

Let's set the initial guesses and bounds accordingly. Note the requirement for a Jovian input radius.

[88]:
```
# define list of BrG line luminosities
n_BrG_list = 23 # desired number of points in grid
BrG_list = np.linspace(L_BrG_estimate-5, L_BrG_estimate+5, n_BrG_list)
# during MCMC, sampled values will be interpolated from the grid defined above for␣
↪faster calculation.

ini_guess = (3100., 3.5, 1.3, 1.6, 3.1, np.median(BrG_list))
```

(continues on next page)

(continued from previous page)

```
bounds = {'Teff':(2000,4000),
          'logg':(2.5,5.5),
          'R':(0.1,5),
          'Av':(0.,5),
          'Rv':(1.,6),
          'BrG': (BrG_min,BrG_max)}
```

Then provide details for each of the line(s), and add an entry in the `em_lines` and `em_grid` dictionaries for each spectral line you'd like to include in the model. Lines can be included in the model without necessarily fitting for their flux with MCMC (e.g. for a known emission line luminosity to be included in the model):

[89]:
```
em_lines = {}
em_lines['BrG'] = (2.1667, input_unit, None) # WL, unit and flux (if known). Flux should␣
↪be None if to be sampled by MCMC.

em_grid = {}
em_grid['BrG'] = BrG_list.copy()
```

Let's set Gaussian priors for R_V in a dictionary. By default, uniform priors are considered for parameters not mentioned in this dictionary. Be sure to use the same dictionary labels as for parameter `labels`.

[90]:
```
priors = {'logg':(3.5,0.2),
          'Rv':(3.1,0.2)}
```

Let's compile all model-related parameters defined above into a dictionary:

[91]:
```
model_params={'grid_param_list':grid_list,
              'model_reader':bts_reader,
              'labels':labels,
              'initial_state':ini_guess,
              'bounds':bounds,
              'em_lines':em_lines,
              'em_grid':em_grid,
              'priors':priors,
              'units_mod':units_mod}
```

In order for the MCMC sampler to run faster, all models of the grid are first resampled (at the same sampling as the measured spectrum), such that only light spectra files are opened and interpolated during the sampling. To do so, set `resamp_before=True` and provide a path and name where the resampled grid will be saved:

[92]:
```
grid_name = 'bts_BrG_resamp_grid.fits'
rel_path = '../static/bts_output/'
output_dir = str((par_path / rel_path).resolve())+'/'
resamp_before = True
```

Important: if a file named `output_dir+grid_name` already exists it will be used, instead of creating a new resmpled grid (this is to save time when a given MCMC sampling is performed multiple times with different options). Make sure to change either `output_dir` or `grid_name` when considering different models or a different range of parameter values in the grid (i.e. different `grid_list`).

We also set a filename where the MCMC results will be written as a pickle, such that they can be accessed without having to run again the MCMC.

[93]:
```
output_file = 'MCMC_results_BrG' # also written in output_dir

output_params = {'resamp_before':resamp_before,
                 'grid_name':grid_name,
                 'output_dir': output_dir,
                 'output_file': output_file}
```

Finally, given the larger number of parameters, we increase the number of iterations and walkers:

[94]:
```
a=2.0
nwalkers=250
niteration_min=100
niteration_limit=1500 # recommended>> 1000
nproc = 2             # number of CPUs

mcmc_params = {'a':a,
               'nwalkers':nwalkers,
               'niteration_min':niteration_min,
               'niteration_limit':niteration_limit,
               'nproc':nproc}
```

Now let's run the MCMC sampler (**Warning: the next cell may take a few minutes to complete**):

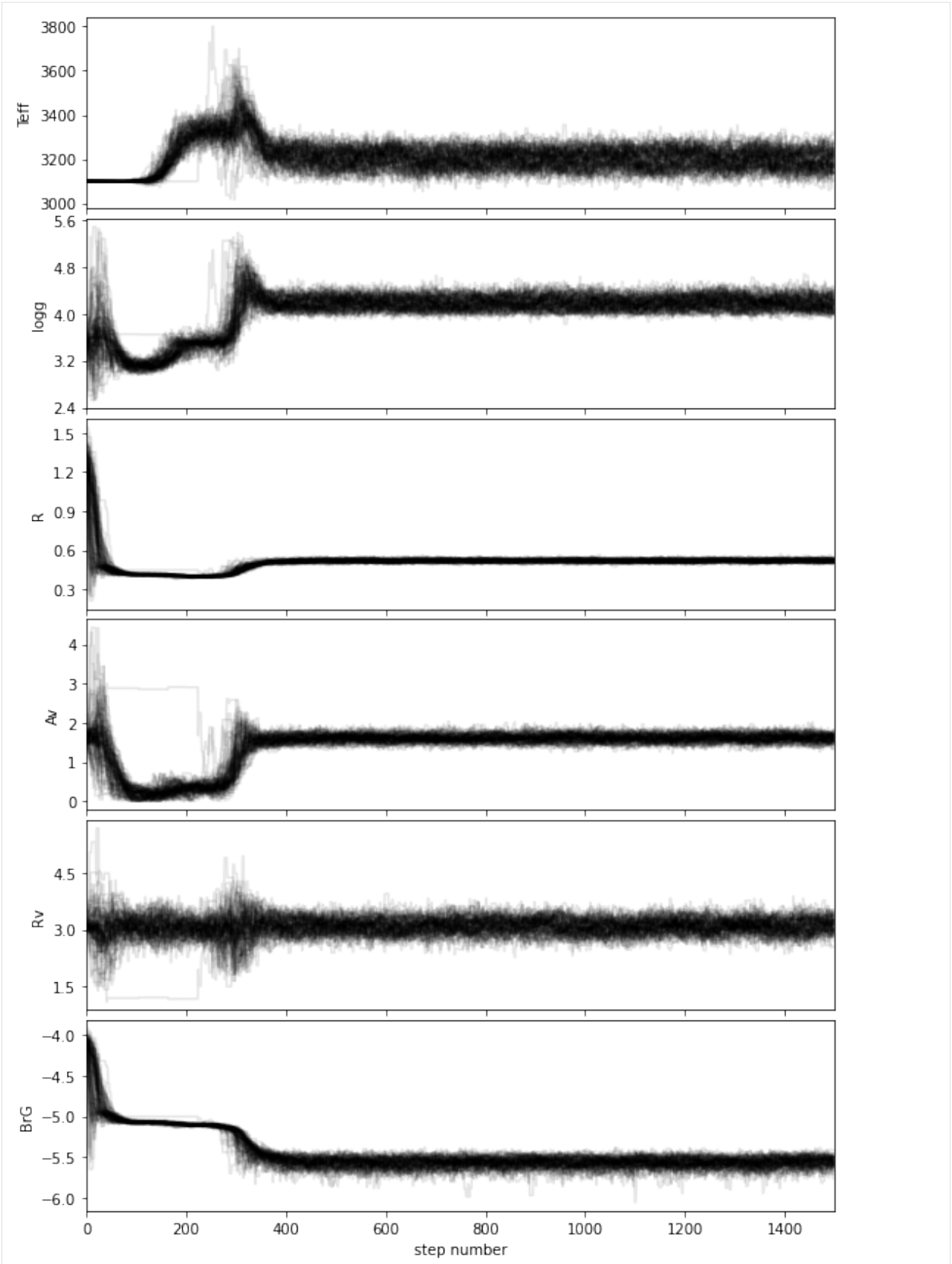
[95]:
```
from special.mcmc_sampling import mcmc_spec_sampling

res = mcmc_spec_sampling(lbda, spec, spec_err, d_st, dlbda_obs=dlbda, **instru_params,
                         **mcmc_params, **conv_params, **model_params, **output_params)
```

```
Fits HDU-0 data successfully loaded. Data shape: (21, 7, 23, 42, 2)
```

[96]:
```
final_chain, ln_proba = res
if star:
    idx_R = labels.index('R')
    final_chain[:,:,idx_R] = final_chain[:,:,idx_R].copy()*conv_RS
    # change units of radius
    units = list(units)
    units[-1] = r'$R_{\odot}$'
    units = tuple(units)
```

Let's inspect the walk plot:

[97]:
```
from special.mcmc_sampling import show_walk_plot

show_walk_plot(final_chain, labels)#, save=False,# output_dir='../datasets/',
```


Teff
3800
3600
3400
3200
3000
logg
5.6
4.8
4.0
3.2
2.4
R
1.5
1.2
0.9
0.6
0.3
Av
4
3
2
1
0
Rv
4.5
3.0
1.5
BrG
−4.0
−4.5
−5.0
−5.5
−6.0
0
200
400
600
800
1000
1200
1400
step number

If you wish to save the walk plot in a pdf, set `save=True` and provide a value for the output directory (`output_dir`).

Based on the walk plot, let's define a conservative burnin factor of 0.5 and apply it to the chain:

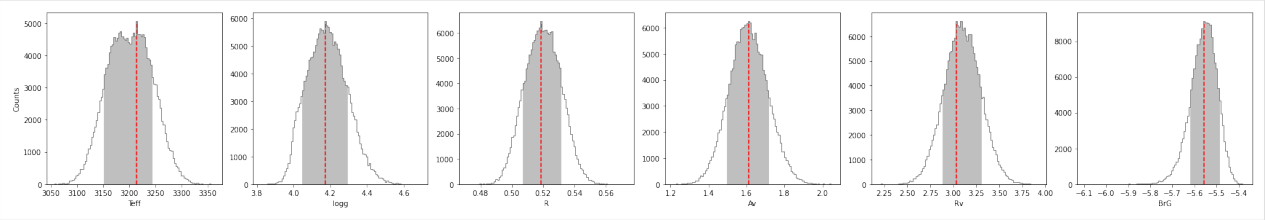
[98]:
```
burnin=0.5
```

[99]:
```
cutoff = int(final_chain.shape[1]//(1/burnin))
ngood_steps = final_chain.shape[1]-cutoff
samples_flat_BrG = final_chain[:, cutoff:, :].reshape((-1,npar))
```

Let's compute the 68.27% confidence intervals on the posterior distribution, after burnin:

[100]:
```
from special.mcmc_sampling import confidence

bins = 100
vals, err = confidence(samples_flat_BrG, labels, cfd=68.27, bins=bins, gaussian_
↪fit=False, weights=None,
                           verbose=True, save=False, bounds=bounds, priors=None)
```

```
percentage for Teff: 70.03626666666666%
percentage for logg: 69.91200000000002%
percentage for R: 69.87893333333332%
percentage for Av: 69.43573333333333%
percentage for Rv: 68.2736%
percentage for BrG: 70.11466666666666%
******* Results for Teff *****


Confidence intervals:
Teff: 3214.0617331626163 [-62.31713260605284,28.8786712076826]
******* Results for logg *****


Confidence intervals:
logg: 4.172630088740946 [-0.1255693907007469,0.11690943272138554]
******* Results for R *****


Confidence intervals:
R: 0.5185647316546919 [-0.011700883183572208,0.012718351286491347]
******* Results for Av *****


Confidence intervals:
Av: 1.6110630569440505 [-0.11327637237633703,0.10488552997808998]
******* Results for Rv *****


Confidence intervals:
Rv: 3.033495340047154 [-0.14717938504144934,0.26838593742852535]
******* Results for BrG *****


Confidence intervals:
BrG: -5.557861370673404 [-0.061250222520235376,0.0684561310520273]
```



If you wish to save the results in a text file, set `save=True` and provide a value for the output directory (`output_dir`).

Let's define the most likely parameters along with their uncertainties in a numpy array:

[101]:
```
post_params = np.zeros([npar,3])
for i in range(npar):
    post_params[i,0] = vals[labels[i]]
    post_params[i,1] = err[labels[i]][0]
    post_params[i,2] = err[labels[i]][1]
```

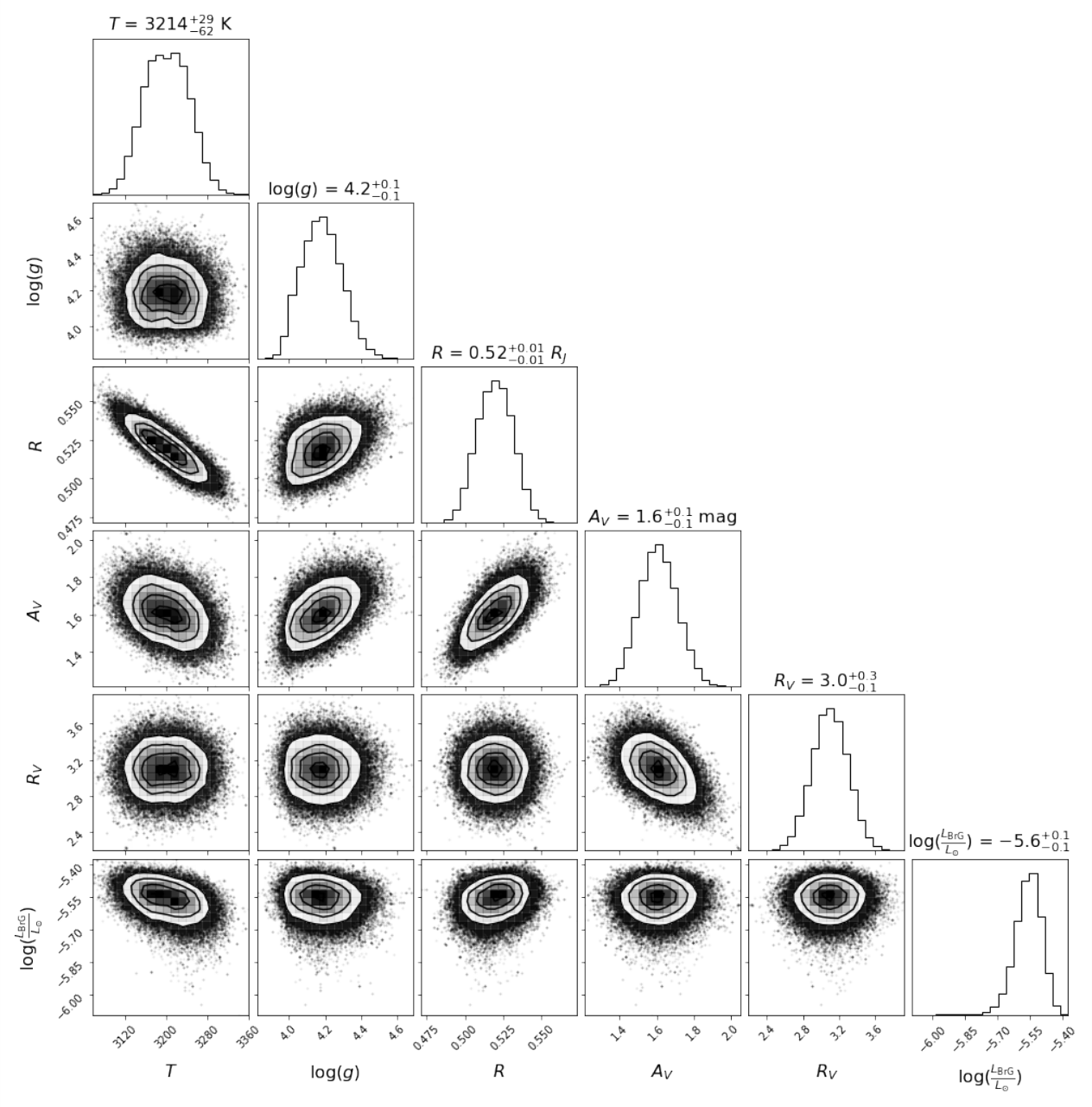
Now, let's have a look at the corner plot. A couple of parameters can be fine tuned for aesthetics, e.g.:

- number of significant digits for each parameter (`ndig`),
- labels to be used in the plot for each parameter (`labels_plot` can be different to the string used in `labels` but `labels` is used if not provided),
- font attributes for plot title and label,
- number of bins to consider for the corner plot histograms.

[102]:
```
ndig = (0,1,2,1,1,1) # should have same length as labels
labels_plot = (r'$T$', r'log($g$)', r'$R$',  r'$A_V$',  r'$R_V$', r"$\log(\frac{L_{\rm␣
↪BrG}}{L_{\odot}})$")
title_kwargs={"fontsize": 16}
label_kwargs={"fontsize": 16}
corner_bins = 20
```

If you wish to save the corner plot in pdf format, set `save=True` and provide a value to `plot_name`.

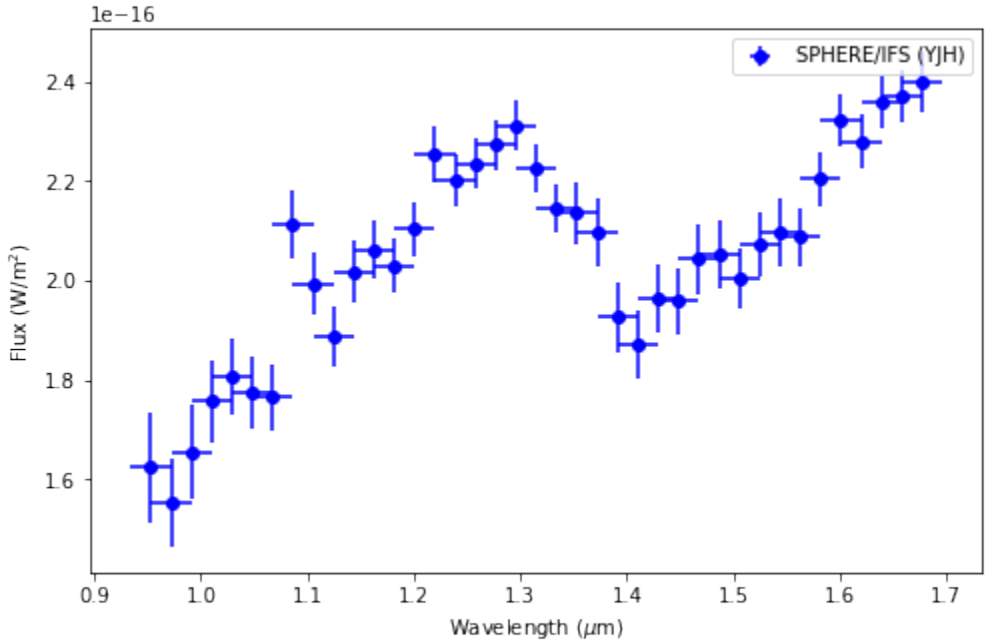
[103]:
```
from special.mcmc_sampling import show_corner_plot

#note: provide the chain before burnin, since burnin is done internally here:
show_corner_plot(final_chain, burnin=burnin, mcmc_res=post_params,
                 units=units, ndig=ndig, labels_plot=labels_plot,
                 labels=labels, #save=True, plot_name='corner_plot.pdf',
                 title_kwargs=title_kwargs, label_kwargs=label_kwargs, bins=corner_bins)
```



Compared to the results obtained without the inclusion of a potential BrG line (affecting the K1 photometry of the point source), we now get a lower value of log(g), more consistent with a young object. Nonetheless, the required BrG line luminosity is significant. This would likely imply the presence of more H recombination lines - which may or may not be present near ~1.09 and ~1.22 μm (see below).

[104]:
```
fig = plt.figure(figsize=figsize)
plt.errorbar(lbda[:-3], lbda[:-3]*spec[:-3], lbda[:-3]*spec_err[:-3], dlbda[:-3], 'bo',
             label = 'SPHERE/IFS (YJH)')
plt.xlabel(r"Wavelength ($\mu$m)")
plt.ylabel(r"Flux (W/m$^2$)")
plt.legend()
```

[104]:
```
<matplotlib.legend.Legend at 0x7fd0502bb9a0>
```



Let's save the likelihood and parameter values of the most likely model for later (*Sec. 5.2*):

[105]:
```
# favoured model has highest likelihood
max_prob_BTS_BrG = np.nanmax(ln_proba)
idx_max = np.unravel_index(np.nanargmax(ln_proba),shape=ln_proba.shape)
bf_params_BTS_BrG = final_chain[idx_max]
bf_params_BTS_BrG
```

[105]:
```
array([ 3.22149387e+03,  4.17221394e+00,  5.14884278e-01,  1.59929704e+00,
        3.05956541e+00, -5.57878381e+00])
```

Go to the top

CHAPTER

NINE

5. COMPARISON OF RESULTS

9.1 5.1. Akaike Information Criterion

For each of the tested type of model above (Secs. 4.2 to 4.5), let's compute the Akaike Information Criterion (AIC) in order to assess which one is likely the best model for our observations. The 4 tested models are:

- Blackbody (2 parameters);
- BT-SETTL + extinction (4 parameters);
- BT-SETTL + 2BB + extinction (8 parameters);
- BT-SETTL + BrG + extinction (6 parameters)

For each type of model, we need the maximum likelihood model among all MCMC samples, and the number of free parameters.

[106]:
```
nparam_list = [2, 4, 6, 6]
max_prob_list = [max_prob_BB, max_prob_BTS, max_prob_BTS_BB, max_prob_BTS_BrG]

n_models = len(nparam_list)
```

[107]:
```
from special.utils_spec import akaike

aic_list = [akaike(max_prob_list[i],nparam_list[i]) for i in range(n_models)]
aic_list
```

[107]:
```
[297.15318710404495, 166.65696588450723, 138.64392704392702, 156.765131720631]
```

Now let's compute the Δ AIC, i.e. the difference with the smallest AIC value obtained for the different models:

[108]:
```
min_aic = np.amin(aic_list)

daic_list = [aic_list[i]-min_aic for i in range(n_models)]
daic_list
```

[108]:
```
[158.50926006011792, 28.013038840580208, 0.0, 18.121204676703968]
```

The values above suggest that the most likely types of model that we considered are the BT-SETTL alone, the BT-SETTL+BB and BT-SETTL+BrG line models.

Since the difference is >>10 for the BB model, we can conclude that there is no support for that kind of model (Burnham & Anderson 2002).

Go to the top

9.2 5.2. Best-fit models

Let's first assemble all the relevant parameters from the different models considered above in different lists:

[109]:
```
lab_models = ['BB', 'BT-SETTL', 'BT-SETTL + BB', 'BT-SETTL + BrG line']
bf_params = [bf_params_BB, bf_params_BTS, bf_params_BTS_BB, bf_params_BTS_BrG]
all_samples_flat = [samples_flat_BB, samples_flat_BTS, samples_flat_BTS_BB, samples_flat_
↪BrG]
all_labels = [('Tbb1', 'Rbb1'),
              ('Teff', 'logg', 'R', 'Av'),
              ('Teff', 'logg', 'R', 'Av', 'Tbb1', 'Rbb1'),
              ('Teff', 'logg', 'R', 'Av', 'Rv', 'BrG')]
all_grid_lists = [None, grid_list, grid_list, grid_list]
all_em_lines = [{}, {}, {}, em_lines]
all_em_grids = [{}, {}, {}, em_grid]
all_ndig = [(0,2),
            (0,1,2,1),
            (0,1,2,1,0,1),
            (0,1,2,1,1,1)]
all_units = [('K', r'$R_J$'),
             ('K', '', r'$R_J$', 'mag'),
             ('K', '', r'$R_J$', 'mag', 'K', r'$R_J$'),
             ('K', '', r'$R_J$', 'mag', '', '')]

n_models = len(lab_models)
```

[110]:
```
bf_params
```

[110]:
```
[array([2.53994358e+03, 7.61007960e-01]),
 array([3.27776316e+03, 4.51097245e+00, 5.22860539e-01, 1.77473043e+00]),
 array([3.15611462e+03, 4.00105754e+00, 4.59765800e-01, 8.45928708e-01,
        1.66216448e+03, 8.28073620e-01]),
 array([ 3.22149387e+03,  4.17221394e+00,  5.14884278e-01,  1.59929704e+00,
         3.05956541e+00, -5.57878381e+00])]
```

Now for each type of model considered above, let's now generate the spectrum corresponding to the most likely parameters inferred with the MCMC. By providing `lbda_obs` to `make_model_from_params`, the returned model will be resampled to match the spectral resolution of the measured spectrum. If provided, the instrumental resolving power and/or photometric filter(s) will also be used. If `lbda_obs` is left to None, the model is returned at the native resolution of the grid used as input. Let's leverage this to have both high-res and resampled spectra.

[111]:
```
from special.model_resampling import make_model_from_params

bf_models = []
bf_models_hr = []

for nn in range(n_models):
    bf_params_nn = tuple(bf_params[nn])
    lbda_model, flux_model = make_model_from_params(bf_params_nn,
                                                    all_labels[nn],
                                                    all_grid_lists[nn],
                                                    d_st,
                                                    model_reader=bts_reader,
```

(continues on next page)

(continued from previous page)

```
                                                em_lines=all_em_lines[nn],
                                                em_grid=all_em_grids[nn],
                                                units_obs=units_obs,
                                                units_mod=units_mod,
                                                interp_order=1,
                                                lbda_obs=lbda,
                                                dlbda_obs=dlbda,
                                                instru_res=instru_res,
                                                instru_idx=instru_idx,
                                                filter_reader=filter_reader)
    bf_models.append([lbda_model, flux_model])
    if nn > 0:
        lbda_model_hr, flux_model_hr = make_model_from_params(bf_params_nn,
                                                              all_labels[nn],
                                                              all_grid_lists[nn],
                                                              d_st,
                                                              model_reader=bts_reader,
                                                              em_lines=all_em_lines[nn],
                                                              em_grid=all_em_grids[nn],
                                                              units_obs=units_obs,
                                                              units_mod=units_mod,
                                                              interp_order=1)
        bf_models_hr.append([lbda_model_hr, flux_model_hr])
```

For models not based on a grid, `lbda_obs` is a mandatory parameter, hence the avoidance of the BB model for high-res models. Let's use the output `lbda_model_hr` from BT-SETTL models, to add a high-res spectrum for the BB model as well.

[112]:
```
lbda_model_hr, flux_model_hr = make_model_from_params(tuple(bf_params[0]),
                                                      all_labels[0],
                                                      all_grid_lists[0],
                                                      d_st,
                                                      model_reader=bts_reader,
                                                      em_lines=all_em_lines[0],
                                                      em_grid=all_em_grids[0],
                                                      units_obs=units_obs,
                                                      units_mod=units_mod,
                                                      interp_order=1,
                                                      lbda_obs=bf_models_hr[-1][0])
bf_models_hr = [[lbda_model_hr, flux_model_hr]]+bf_models_hr
```

Now crop the model spectra to the same wavelength range as the measured spectrum:

[113]:
```
from special.utils_spec import find_nearest

for nn in range(n_models):
    lbda_model_hr, flux_model_hr = bf_models_hr[nn]
    # LOAD BEST FIT MODEL
    idx_ini = find_nearest(lbda_model_hr,0.99*lbda[0], constraint='floor')
    idx_fin = find_nearest(lbda_model_hr,1.01*lbda[-1], constraint='ceil')
    lbda_model_hr = lbda_model_hr[idx_ini:idx_fin+1]
    flux_model_hr = flux_model_hr[idx_ini:idx_fin+1]
```

(continues on next page)

(continued from previous page)

```
    dlbda_hr = np.mean(lbda_model_hr[1:]-lbda_model_hr[:-1])
    bf_models_hr[nn] = [lbda_model_hr, flux_model_hr]
```

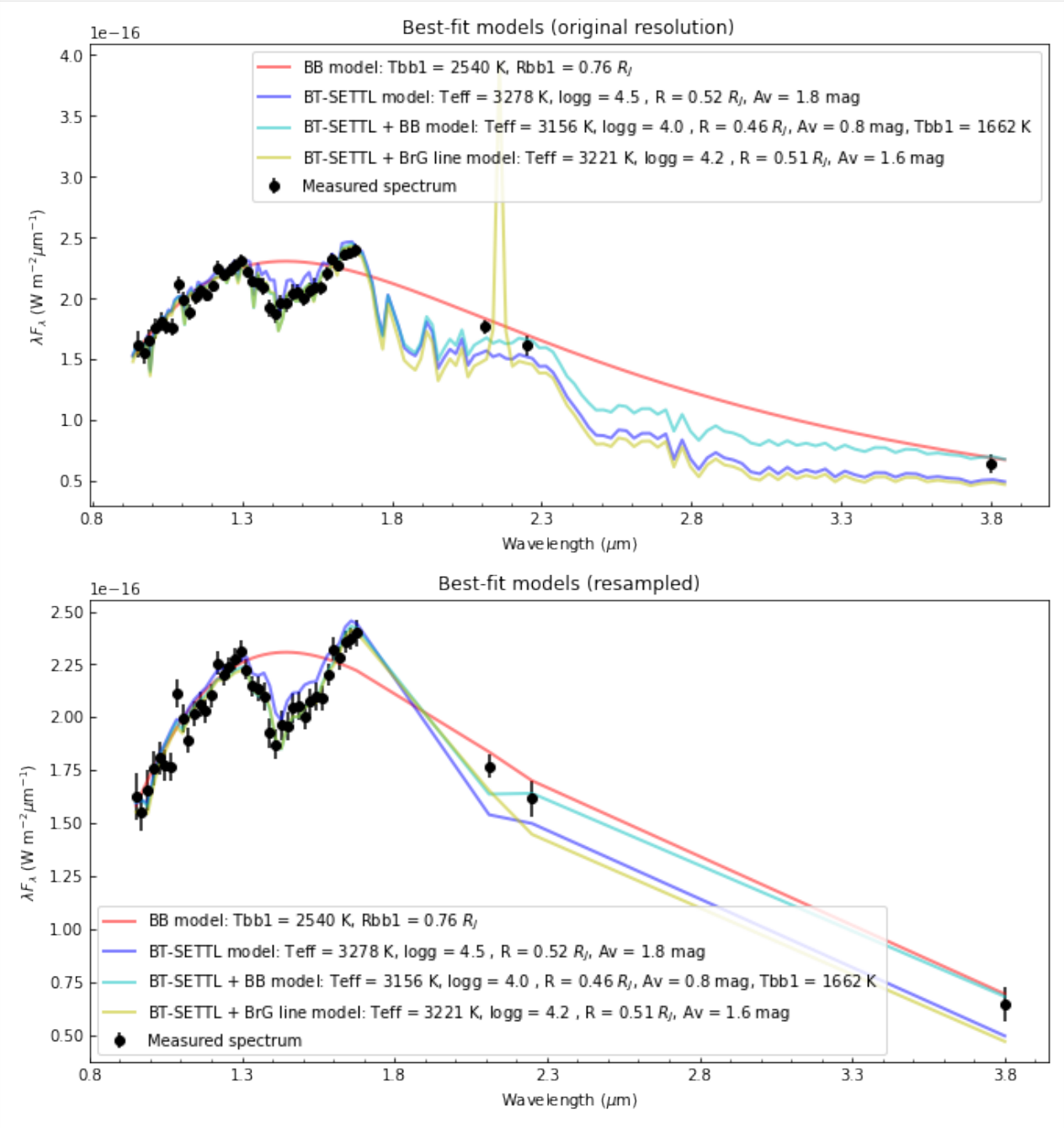
Now let's plot the best-fit models at their native resolution (first panel) and after resampling (second panel):

[114]:
```
fig, axes = plt.subplots(2,1,figsize=(11,12))

# plot options
cols = ['k', 'r', 'b', 'c', 'y', 'm'] # colors of different models
maj_tick_sp = 0.5 # WL spacing for major ticks
min_tick_sp = maj_tick_sp / 5. # WL spacing for minor ticks

# Titles
axes[0].set_title("Best-fit models (original resolution)")
axes[1].set_title("Best-fit models (resampled)")

# Plot measured spectrum
axes[0].errorbar(lbda, lbda*spec,
             lbda*spec_err, fmt=cols[0]+'o',
             label='Measured spectrum')
axes[1].errorbar(lbda, lbda*spec,
             lbda*spec_err, fmt=cols[0]+'o',
             label='Measured spectrum')

# Set axes labels
axes[0].set_xlabel(r"Wavelength ($\mu$m)")
axes[0].set_ylabel(r"$\lambda F_{\lambda}$ (W m$^{-2} \mu$m$^{-1}$)")
axes[1].set_xlabel(r"Wavelength ($\mu$m)")
axes[1].set_ylabel(r"$\lambda F_{\lambda}$ (W m$^{-2} \mu$m$^{-1}$)")

# Plot best-fit models
for i in range(2):
    if i == 0:
        bf_models_i = bf_models_hr
    else:
        bf_models_i = bf_models
    for nn in range(n_models):
        lbda_model, flux_model = bf_models_i[nn]

        lab_str = '{0} = {1:.{2}f} {3}'
        if nn==0 or nn ==1:
            show_labs = range(len(bf_params[nn]))
        elif nn==2:
            show_labs = range(len(bf_params[nn])-1)
        else:
            show_labs = list(range(len(bf_params[nn])-2))
        lab_str_list = [lab_str.format(all_labels[nn][j], bf_params[nn][j], all_
↪ndig[nn][j], all_units[nn][j]) for j in show_labs]
        sep = ', '
        label = "{} model: {}".format(lab_models[nn], sep.join(lab_str_list))
        axes[i].plot(lbda_model, lbda_model*flux_model, cols[1+nn], linewidth=2,
                 alpha=0.5, label=label)
```

(continues on next page)

(continued from previous page)

```
        min_tick = lbda[0]-dlbda[0]/2-((lbda[0]-dlbda[0]/2)%0.2)
        max_tick = lbda[-1]+dlbda[-1]/2+(0.2-((lbda[-1]+dlbda[-1]/2)%0.2))
        major_ticks1 = np.arange(min_tick,max_tick,maj_tick_sp)
        minor_ticks1 = np.arange(min_tick,max_tick,min_tick_sp)

        axes[i].set_xticks(major_ticks1)
        axes[i].set_xticks(minor_ticks1, minor = True)
        axes[i].tick_params(which = 'both', direction = 'in')
        axes[i].legend(loc='best')

plt.show()
```



Visually, it looks like a combination of a $T_{\rm eff} \sim 3200$ K BT-SETTL model, an extra black-body component ($T_{\rm eff} \sim 1700$ K) and several emission lines may be able to reproduce the entire observed spectrum. However, one has to be cautious of not overfitting the data …

Go to the top

9.3 5.3. Models from the posterior distribution

It can be useful to plot a large number of models randomly picked from the posterior distribution in order to have an idea of the uncertainty in different parts of the spectrum. Let's consider 200 samples from the posterior distribution:

[115]:
```
n_samp = 200
```

[116]:
```
import random

all_plot_samples = []
for nn in range(n_models):
    print("################## Model {} ##################".format(lab_models[nn]))
    samp_flux = []
    lbda_samp = None # use native model resolution (set to `lbda` for measured spectrum␣
↪sampling)
    if nn == 0:
        # set the wavelength sampling for
        lbda_samp=bf_models_hr[-1][0]
    for ii in range(n_samp):
        # draw a random integer
        idx = random.randint(0, len(all_samples_flat[nn]))
        param_samp = tuple(all_samples_flat[nn][idx])
        samp_lbda, tmp = make_model_from_params(param_samp, all_labels[nn], all_grid_
↪lists[nn], d_st,
                                                model_reader=bts_reader, em_lines=all_em_
↪lines[nn],
                                                em_grid=all_em_grids[nn], units_
↪obs=units_obs,
                                                units_mod=units_mod, interp_order=1,␣
↪lbda_obs=lbda_samp)

        samp_flux.append(tmp)
    # Crop to same range as measurements
    idx_ini = find_nearest(samp_lbda,0.99*lbda[0], constraint='floor')
    idx_fin = find_nearest(samp_lbda,1.01*lbda[-1], constraint='ceil')
    samp_lbda = samp_lbda[idx_ini:idx_fin+1]
    samp_fluxes = [samp_flux[ii][idx_ini:idx_fin+1] for ii in range(n_samp)]
    all_plot_samples.append([samp_lbda,samp_fluxes])
```

```
################## Model BB ##################
################## Model BT-SETTL ##################
################## Model BT-SETTL + BB ##################
################## Model BT-SETTL + BrG line ##################
```

```
/Users/valentin/GitHub/special/special/model_resampling.py:284: RuntimeWarning: overflow␣
↪encountered in power
  flux_ratio_ext = np.power(10., -extinc_curve/2.5)
```

Now let's plot the samples:

[117]:
```
fig, axes = plt.subplots(4,1,figsize=(11,24))

# plot options
cols = ['k', 'r', 'b', 'c', 'y', 'm'] # colors of different models
```

(continues on next page)

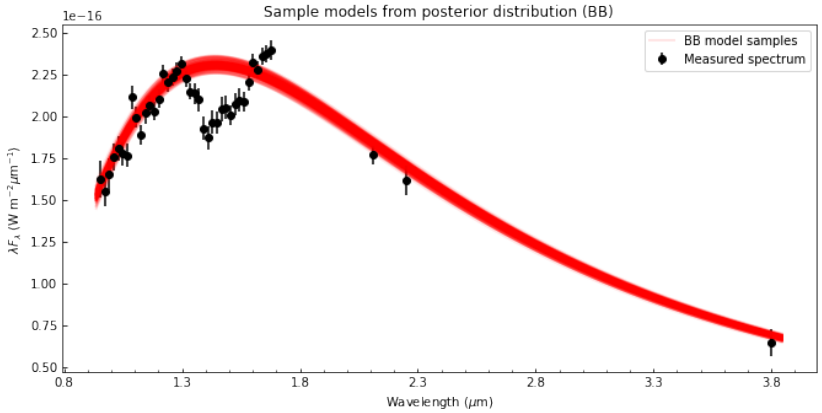
(continued from previous page)

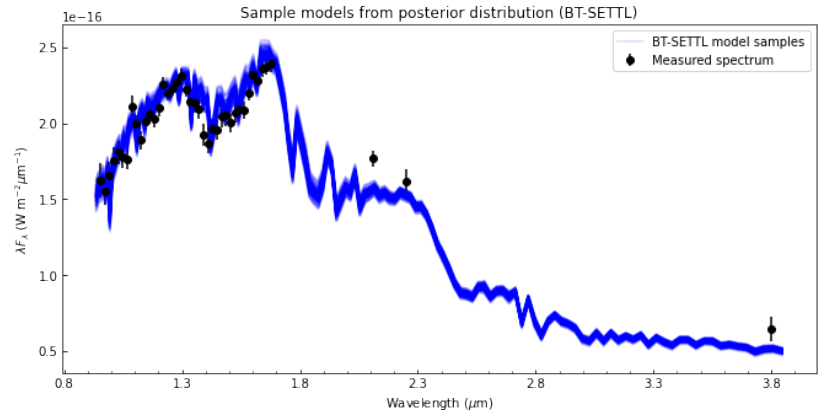
```
maj_tick_sp = 0.5 # WL spacing for major ticks
min_tick_sp = maj_tick_sp / 5. # WL spacing for minor ticks

# Titles
for nn in range(n_models):
    axes[nn].set_title("Sample models from posterior distribution ({})".format(lab_
↪models[nn]))

    # Set axes labels
    axes[nn].set_xlabel(r"Wavelength ($\mu$m)")
    axes[nn].set_ylabel(r"$\lambda F_{\lambda}$ (W m$^{-2} \mu$m$^{-1}$)")

    # Options depending on model
    if nn==0 or nn ==1:
        show_labs = range(len(bf_params[nn]))
    elif nn==2:
        show_labs = range(len(bf_params[nn])-1)
    else:
        show_labs = list(range(len(bf_params[nn])-2))

    # Plot sample models
    lbda_samp, fluxes_model = all_plot_samples[nn]
    for i in range(n_samp):
        flux_model = fluxes_model[i]
        if i == 0:
            lab_str = '{0} = {1:.{2}f} {3}'
            lab_str_list = [lab_str.format(all_labels[nn][j], bf_params[nn][j], all_
↪ndig[nn][j], all_units[nn][j]) for j in show_labs]
            sep = ', '
            label = "{} model samples".format(lab_models[nn])#: {}".format(lab_
↪models[nn], sep.join(lab_str_list))
        else:
            label = None
        axes[nn].plot(lbda_samp, lbda_samp*flux_model, cols[1+nn], linewidth=2, alpha=0.
↪1, label=label)

    min_tick = lbda[0]-dlbda[0]/2-((lbda[0]-dlbda[0]/2)%0.2)
    max_tick = lbda[-1]+dlbda[-1]/2+(0.2-((lbda[-1]+dlbda[-1]/2)%0.2))
    major_ticks1 = np.arange(min_tick,max_tick,maj_tick_sp)
    minor_ticks1 = np.arange(min_tick,max_tick,min_tick_sp)

    axes[nn].set_xticks(major_ticks1)
    axes[nn].set_xticks(minor_ticks1, minor = True)
    axes[nn].tick_params(which = 'both', direction = 'in')


    # Plot measured spectrum
    axes[nn].errorbar(lbda, lbda*spec,
                      lbda*spec_err, fmt=cols[0]+'o',
                      label='Measured spectrum')
    axes[nn].legend(loc='best')
```

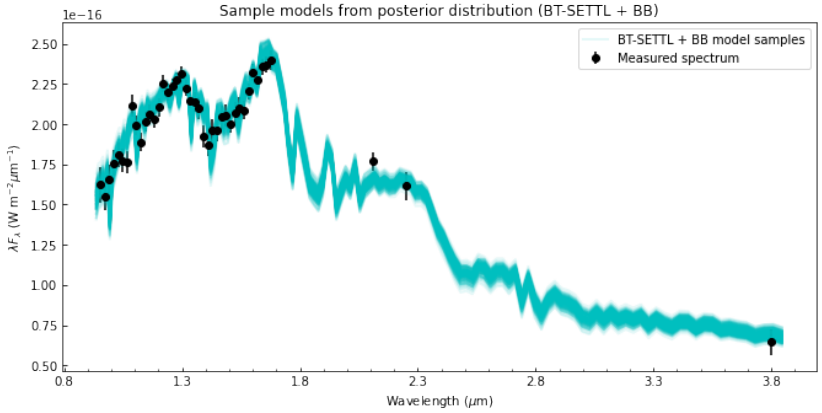
(continues on next page)

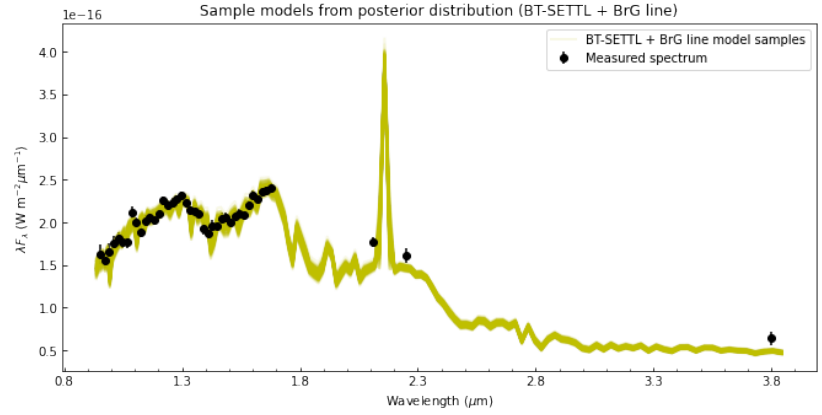
(continued from previous page)

```
#plt.savefig("Sample_models.pdf", bbox_inches='tight')

plt.show()
```


Sample models from posterior distribution (BB)
1e−16
BB model samples
Measured spectrum
λF_λ (W m$^{-2}\mu$m^{-1})
Wavelength (μm)


Sample models from posterior distribution (BT-SETTL)
1e−16
BT-SETTL model samples
Measured spectrum
λF_λ (W m$^{-2}\mu$m^{-1})
Wavelength (μm)


Sample models from posterior distribution (BT-SETTL + BB)
1e−16
BT-SETTL + BB model samples
Measured spectrum
λF_λ (W m$^{-2}\mu$m^{-1})
Wavelength (μm)


Sample models from posterior distribution (BT-SETTL + BrG line)
1e−16
BT-SETTL + BrG line model samples
Measured spectrum
λF_λ (W m$^{-2}\mu$m^{-1})
Wavelength (μm)

Go to the top

CHAPTER

TEN

6. NESTED SAMPLER EXAMPLES

The procedure to set the nested sampler routines is similar to what we saw with the MCMC, as most parameters of `mcmc_spec_sampling` and `nested_spec_sampling` are identical.

The main differences are the `sampler`, `method`, `npoints` or `dlogz` parameters for the nested samplers:

- the `sampler` can be set to `'ultranest'` or `'nestle'`.
- in the case of `'nestle'`, the `method` can be set to `'classic'` (MCMC; Skilling 2004), `'single'` (single ellipsoid; Mutherjee et al. 2006) or `'multi'` (original multinest implementation as in Feroz et al. 2009, with conservative ellipsoid splitting).
- `npoints` corresponds to the number of active live points (minimum number in the case of UltraNest);
- `dlogz` is the target evidence uncertainty. The stopping criterion is log(z + z_est) - log(z) < dlogz. There are additional criteria for UltraNest.

Other sampler-specific parameters (e.g. `decline_factor` and `rstate` for nestle, or `frac_remain` and `dKL` for UltraNest) can be provided as `kwargs`. The interested reader is referred to the documentation of UltraNest and nestle for details on how to appropriately set the values of the different parameters.

Below we show three examples of nested sampler runs, using `nestle` and `ultranest`, with the BT-SETTL grid, considering A_V as a free parameter. Let's first set the parameters as in *Sec. 4.3*:

[118]:
```
from utils import bts_reader

teff_list = np.linspace(2000,4000,num=21).tolist()
logg_list = np.linspace(2.5,5.5,num=7).tolist()
grid_list = [teff_list, logg_list]

ini_guess = (3100., 3.5, 1.3, 1.6)

labels = ('Teff', 'logg', 'R', 'Av')
labels_plot = (r'$T$', r'log($g$)', r'$R$',  r'$A_V$')
units = ('K', '', r'$R_J$', 'mag')
ndig = (0,1,2,1) # number of significant digits
units_mod = 'si'
bounds = {'Teff':(2000,4000),
          'logg':(2.5,5.5),
          'R':(0.1,5),
          'Av':(0.,5)}
model_params={'grid_param_list':grid_list,
              'model_reader':bts_reader,
              'labels':labels,
              'bounds':bounds,
```

(continues on next page)

(continued from previous page)

```
              'units_mod':units_mod}

grid_name='bts_resamp_grid.fits'
rel_path = '../static/bts_output/'
output_dir = str((par_path / rel_path).resolve())+'/'
resamp_before=True
output_params = {'resamp_before':resamp_before,
                 'grid_name':grid_name,
                 'output_dir': output_dir}
```

10.1 6.1. Nestle - single ellipsoid method

Now let's set the parameters that are specific to the nested sampler:

[119]:
```
sampler='nestle'
method='single'
npoints=1500
dlogz=0.1
```

[120]:
```
nested_params = {'sampler':sampler,
                 'method':method,
                 'npoints':npoints,
                 'dlogz':dlogz}
```

Now let's run the nested sampler. **Warning: the next cell may take up to a few minutes to complete** - reduce the number of active points for a faster calculation.

[121]:
```
from special.nested_sampling import nested_spec_sampling

res = nested_spec_sampling(ini_guess, lbda, spec, spec_err, d_st, dlbda_obs=dlbda,␣
↪**instru_params,
                           **nested_params, **model_params, **output_params)
```

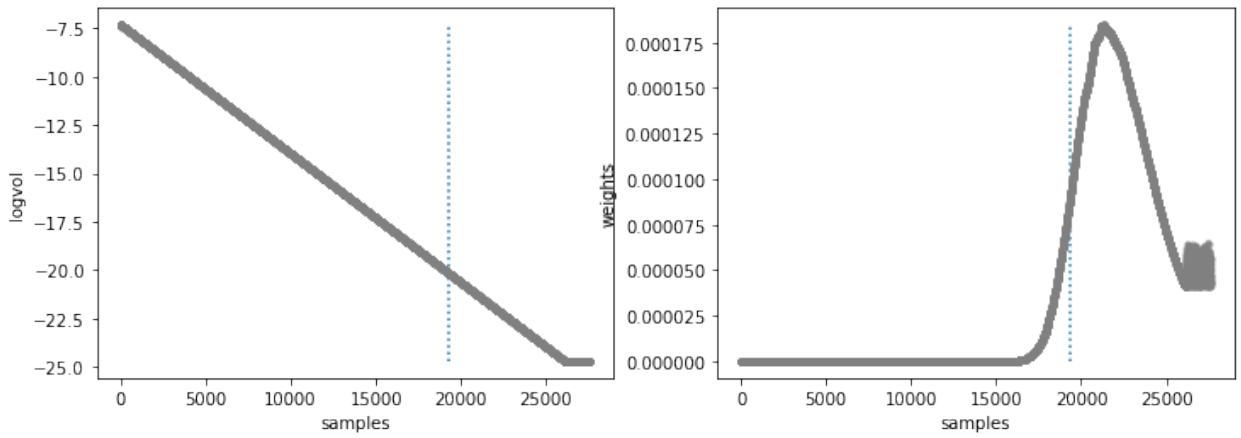
```
Fits HDU-0 data successfully loaded. Data shape: (21, 7, 42, 2)
-------------------------------------------------------------------------------
Starting time: 2023-05-08 16:59:27
-------------------------------------------------------------------------------
Prior bounds on parameters:
Teff [2000,4000]
logg [2.5,5.5]
R [0.1,5]
Av [0.0,5]

Using 1500 active points

Total running time:
Running time:  0:06:05.177341
-------------------------------------------------------------------------------
```

Now let's plot the results:

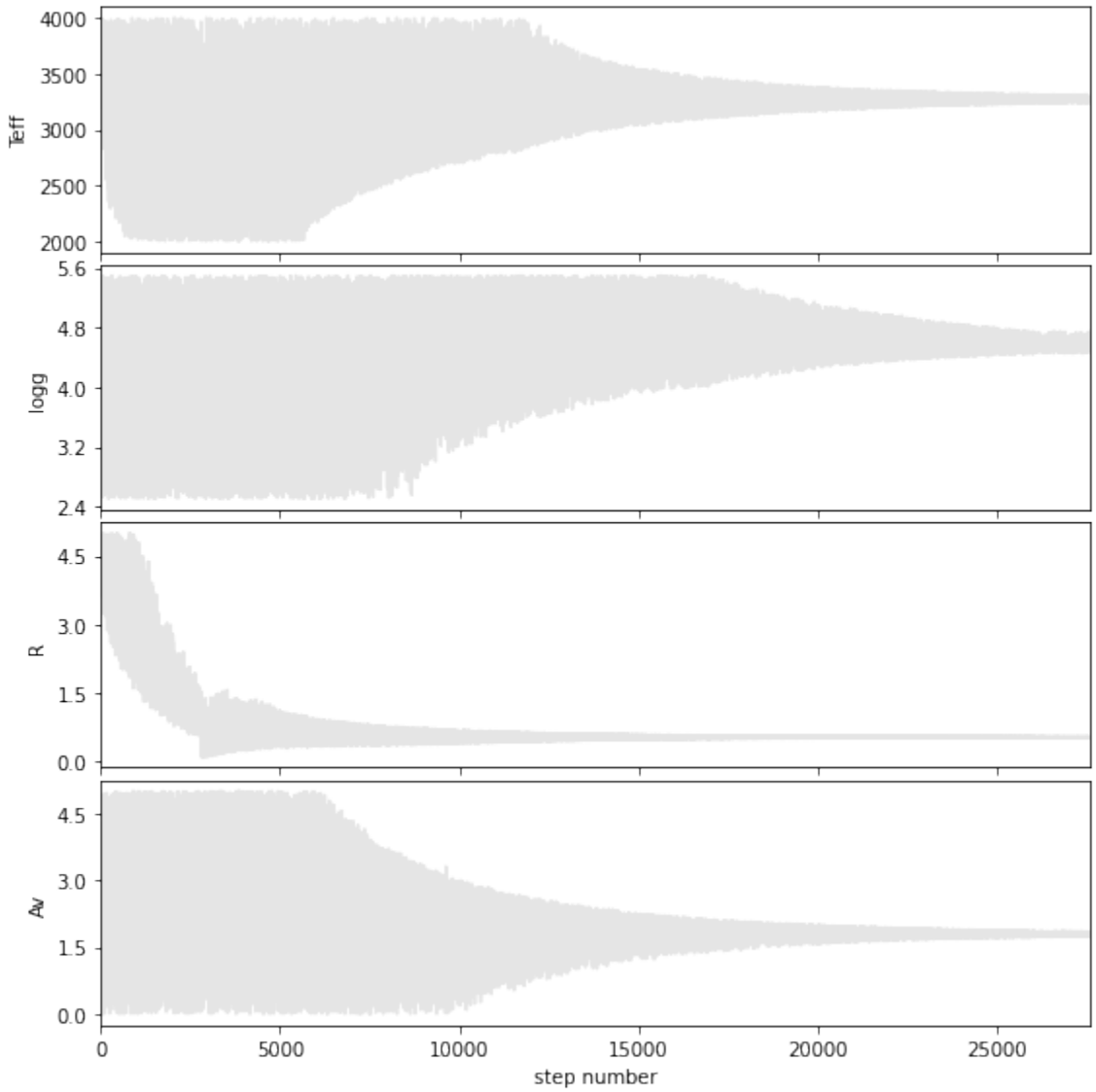
[122]:
```
from special.nested_sampling import show_nestle_results

final_res = show_nestle_results(res, labels, method=method, burnin=0.7, bins=None,␣
↪cfd=68.27,
                                units=units, ndig=ndig, labels_plot=labels_plot,␣
↪save=False,
                                output_dir='./', plot=True)
```

```
niter: 26105
ncall: 95512
nsamples: 27605
logz: -94.391 +/-  0.093
h: 13.064
```

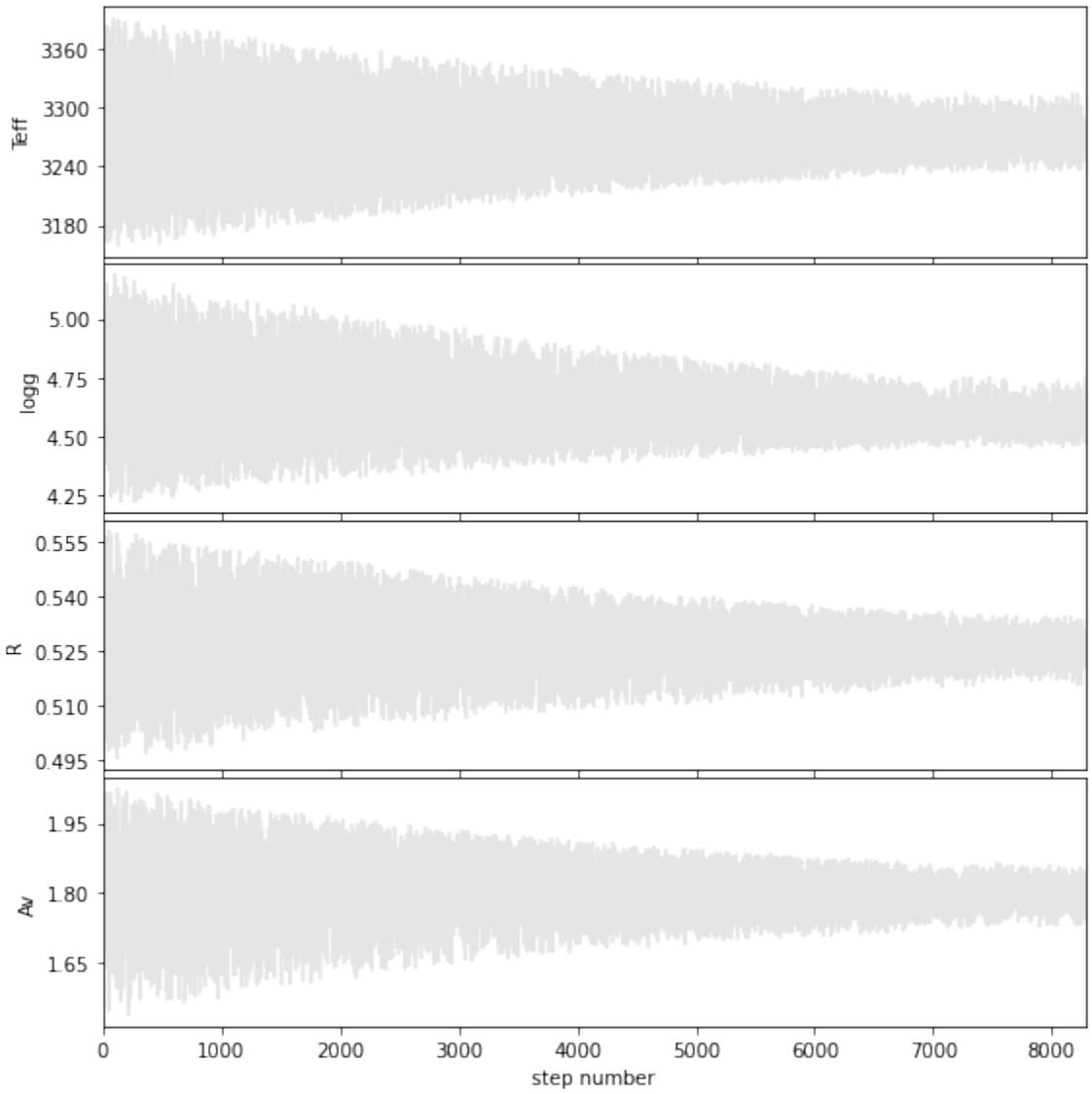
Natural log of prior volume and Weight corresponding to each sample



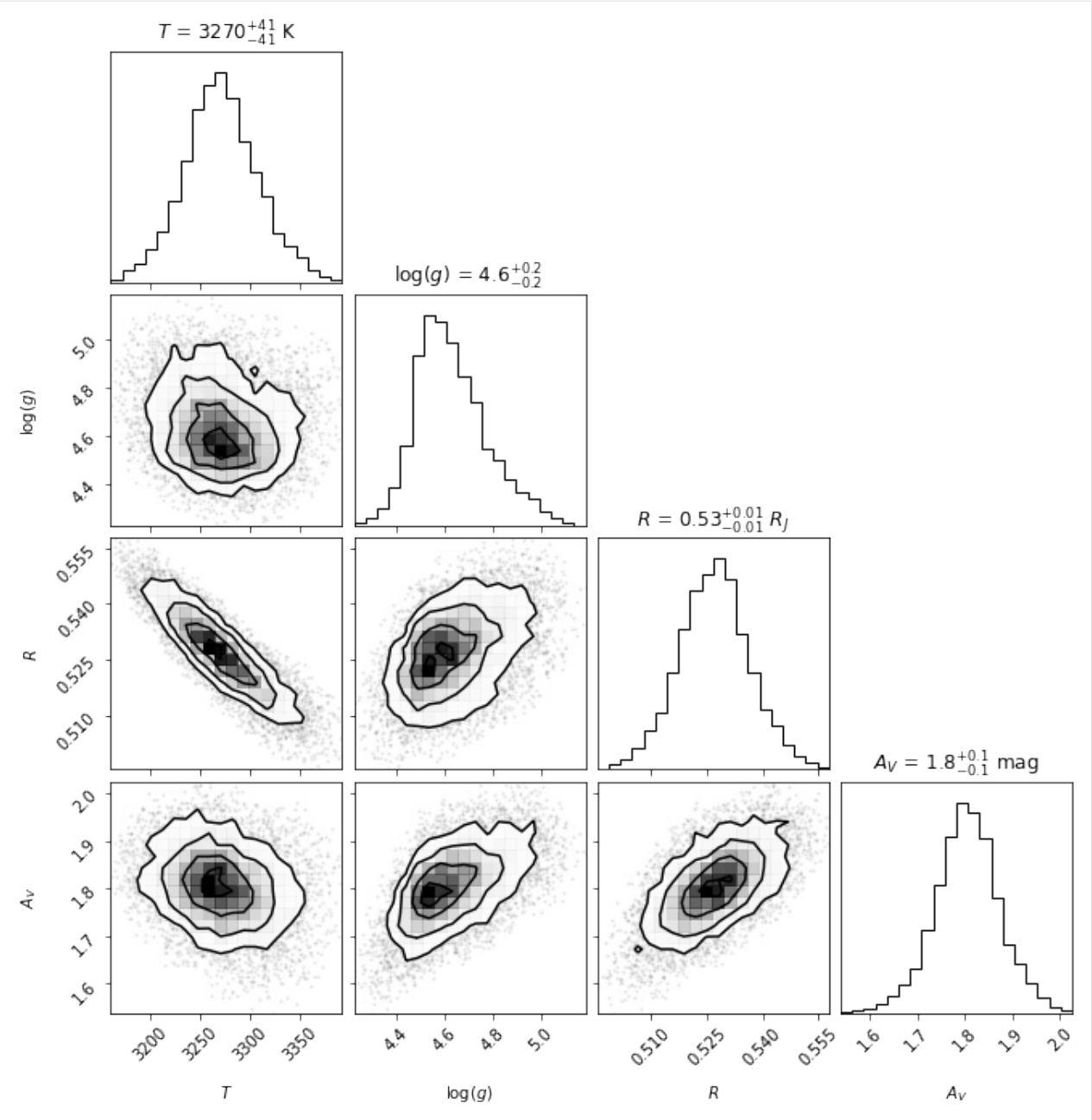
Walk plots before the burnin



Walk plots after the burnin




```
Weighted mean +- sqrt(covariance)
Teff = 3270 +/- 41
logg = 4.6 +/- 0.2
R = 0.53 +/- 0.01
Av = 1.8 +/- 0.1

Hist bins = 91
```

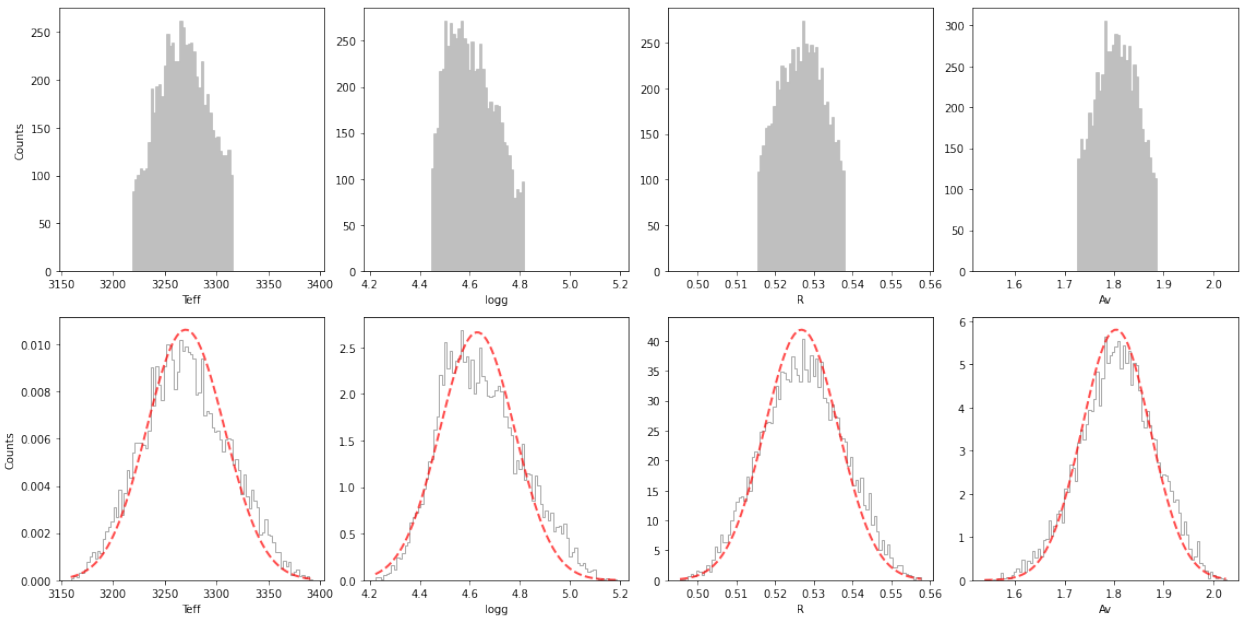



```
Confidence intervals
percentage for Teff: 68.69325372637766%
percentage for logg: 69.00875209816401%
percentage for R: 68.58988627363547%
percentage for Av: 69.12226646174167%
******* Results for Teff *****


Confidence intervals:
Teff: 3265.6231426509084 [-47.32906217538448,49.887389860540225]
Gaussian fit results:
```

(continues on next page)

(continued from previous page)

```
Teff: 3270.232216747061 +-37.552482976221874
******* Results for logg *****


Confidence intervals:
logg: 4.566973411065405 [-0.1219748077333822,0.24925286797691104]
Gaussian fit results:
logg: 4.629821414822097 +-0.1497690355250289
******* Results for R *****


Confidence intervals:
R: 0.52729907409839 [-0.01196794055372663,0.010600175919015076]
Gaussian fit results:
R: 0.5268292282938275 +-0.009526056906226689
******* Results for Av *****


Confidence intervals:
Av: 1.7820752344394402 [-0.05625272255452329,0.10446934188697177]
Gaussian fit results:
Av: 1.8038934043192472 +-0.06874061336251147
```



The best-fit parameters, with uncertainty are:

[123]:
```
for p in range(len(labels)):
    fmt = "{{:.{0}f}}".format(ndig[p]).format
    line = r"{0}: {1}+/-{2} {3}"
    print(line.format(labels[p],fmt(final_res[p][0]),fmt(final_res[p][1]), units[p]))
```

```
Teff: 3270+/-41 K
logg: 4.6+/-0.2
R: 0.53+/-0.01 $R_J$
```

(continues on next page)

(continued from previous page)

```
Av: 1.8+/-0.1 mag
```

10.2 6.2. Nestle - multi-ellipsoid method

Let's adapt the method and number of active points:

[124]:
```
sampler='nestle'
method='multi'
npoints=2000
dlogz=0.5
```

[125]:
```
nested_params = {'sampler':sampler,
                 'method':method,
                 'npoints':npoints,
                 'dlogz':dlogz}
```

Now let's run the nested sampler. **Warning: the next cell may take up to a few minutes to complete** - reduce the number of active points for a faster calculation.

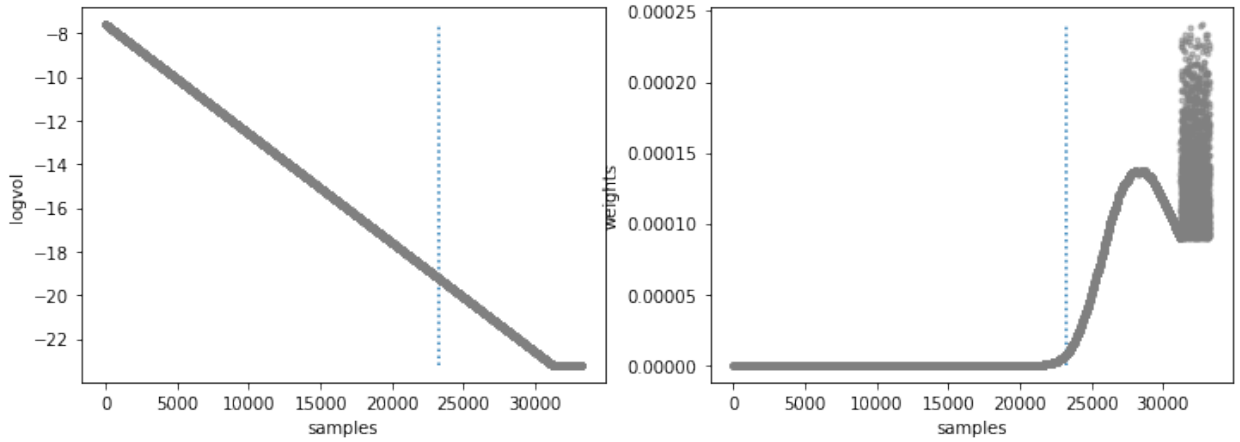
[126]:
```
from special.nested_sampling import nested_spec_sampling

res = nested_spec_sampling(ini_guess, lbda, spec, spec_err, d_st, dlbda_obs=dlbda,␣
↪**instru_params,
                           **nested_params, **model_params, **output_params)
```

```
Fits HDU-0 data successfully loaded. Data shape: (21, 7, 42, 2)
-------------------------------------------------------------------------------
Starting time: 2023-05-08 17:05:34
-------------------------------------------------------------------------------
Prior bounds on parameters:
Teff [2000,4000]
logg [2.5,5.5]
R [0.1,5]
Av [0.0,5]

Using 2000 active points

Total running time:
Running time:  0:07:13.804959
-------------------------------------------------------------------------------
```

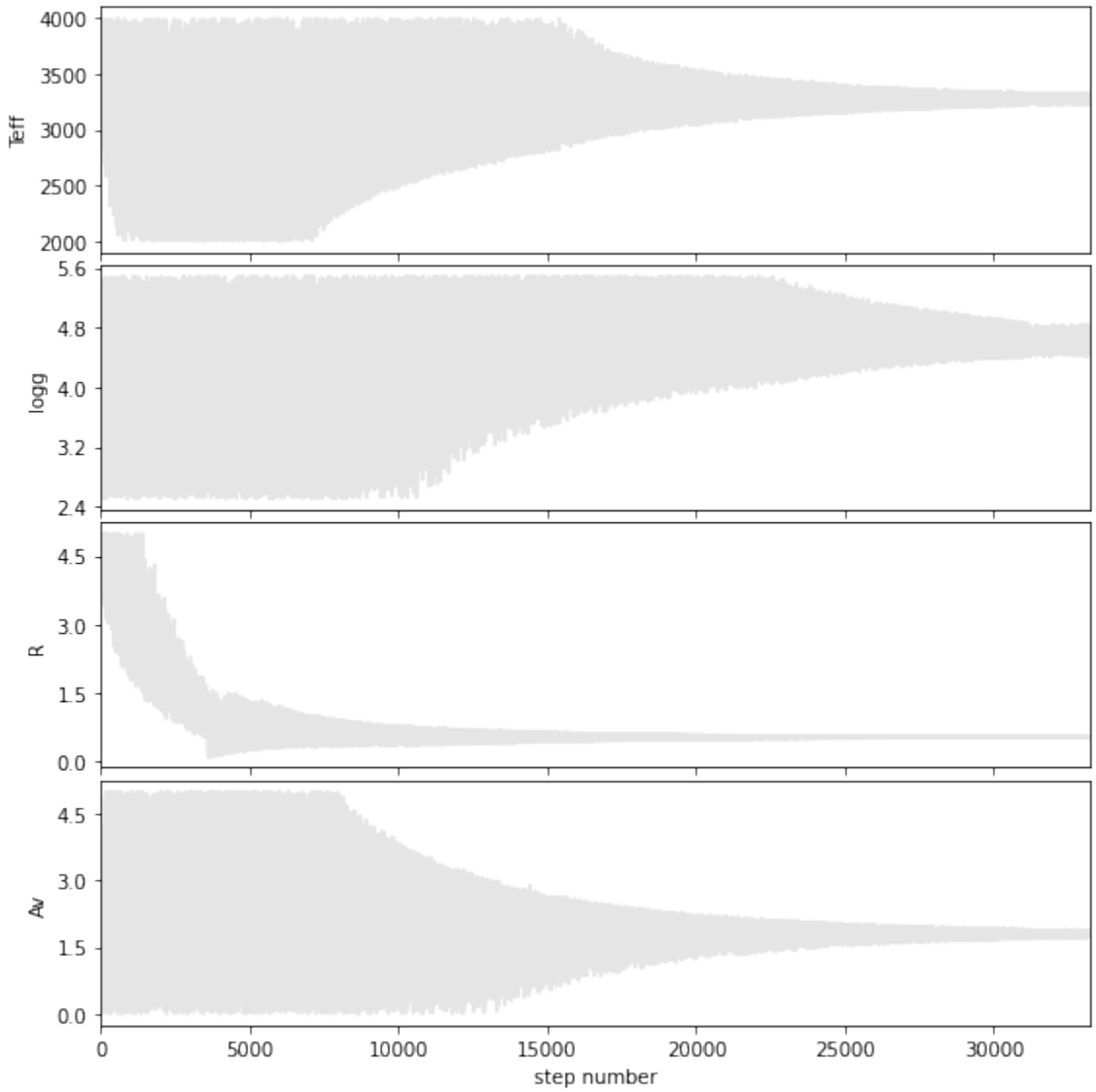
Now let's plot the results:

[127]:
```
from special.nested_sampling import show_nestle_results

final_res = show_nestle_results(res, labels, method=method, burnin=0.7, bins=None,␣
↪cfd=68.27,
                                units=units, ndig=ndig, labels_plot=labels_plot,␣
↪save=False,
                                output_dir='./', plot=True)
```

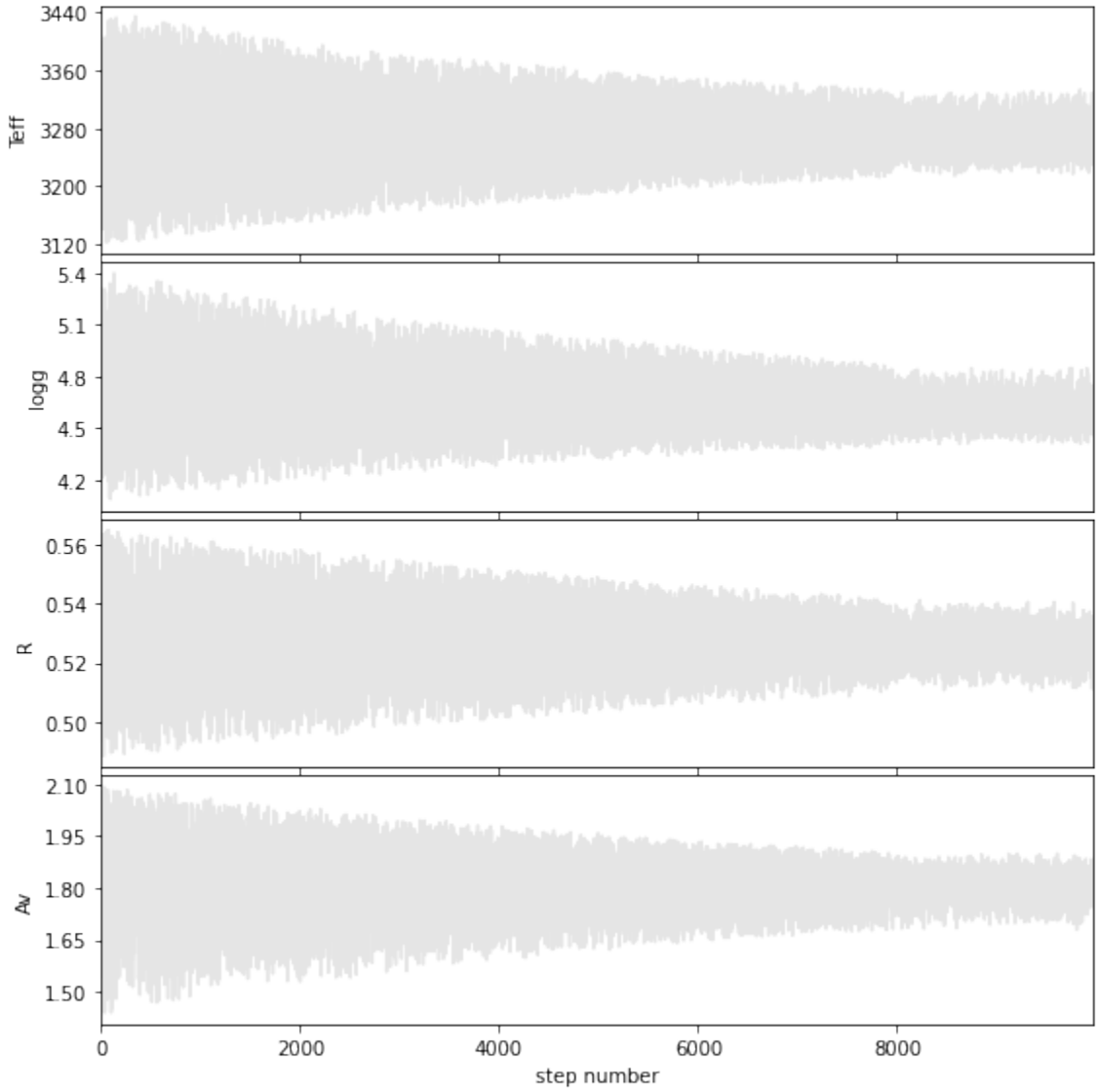
```
niter: 31211
ncall: 110849
nsamples: 33211
logz: -94.207 +/-  0.080
h: 12.896

Natural log of prior volume and Weight corresponding to each sample
```



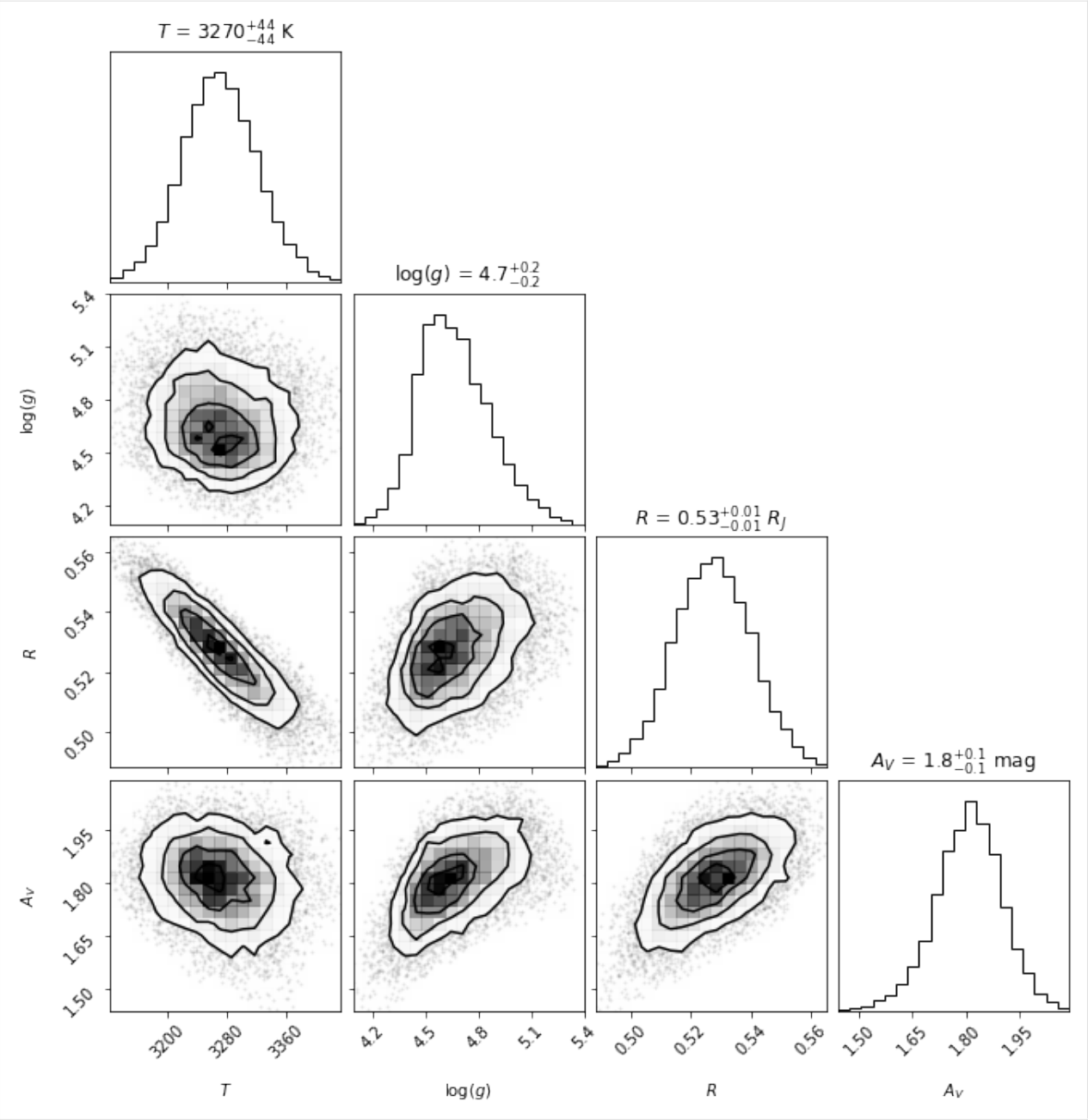

```
Walk plots before the burnin
```



Walk plots after the burnin




```
Weighted mean +- sqrt(covariance)
Teff = 3270 +/- 44
logg = 4.7 +/- 0.2
R = 0.53 +/- 0.01
Av = 1.8 +/- 0.1

Hist bins = 99
```

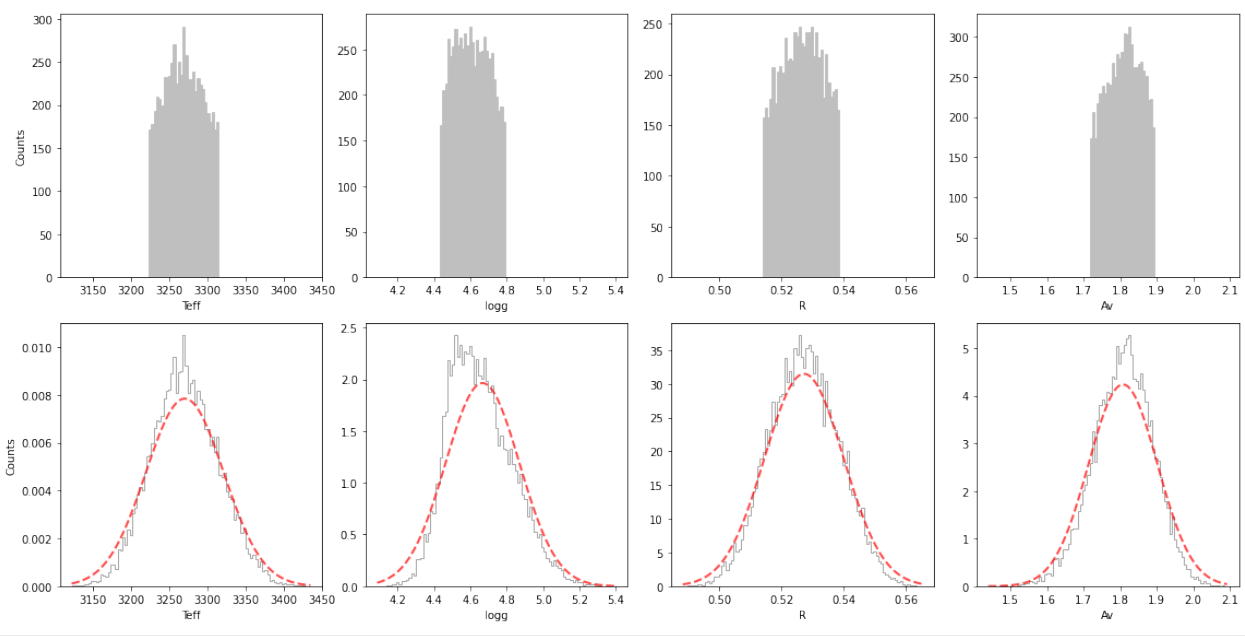



```
Confidence intervals
percentage for Teff: 69.72033635934162%
percentage for logg: 68.38961119720196%
percentage for R: 68.88926242249346%
percentage for Av: 69.4935601917666%
******* Results for Teff *****


Confidence intervals:
Teff: 3269.203121254572 [-45.727599259344515,45.727599259344406]
Gaussian fit results:
```

(continues on next page)

(continued from previous page)

```
Teff: 3270.466306068884 +-50.82823562458199
******* Results for logg *****


Confidence intervals:
logg: 4.518232629957659 [-0.0862444516965839,0.27200173227384017]
Gaussian fit results:
logg: 4.666203996451627 +-0.20315435230105794
******* Results for R *****


Confidence intervals:
R: 0.5259084225783466 [-0.012026642989035552,0.012802555439941132]
Gaussian fit results:
R: 0.5272967777306088 +-0.012667103230549065
******* Results for Av *****


Confidence intervals:
Av: 1.825295179162057 [-0.10888022904255945,0.06928741848162878]
Gaussian fit results:
Av: 1.8077748841054169 +-0.0940790048831327
```



We see that for the case of the spectrum of CrA-9B/b with a BT-SETTL + extinction model, even when we allow for multiple nests, the highest likelihood area in the parameter space still corresponds to an individual ellipsoid.

10.3 6.3. UltraNest

Let's adapt the sampler and number of active points:

[128]:
```
sampler='ultranest'
npoints=1000
dlogz=0.2
```

[129]:
```
nested_params = {'sampler':sampler,
                 'method':method,
                 'npoints':npoints,
                 'dlogz':dlogz}
```

Now let's run the nested sampler. **Warning: the next cell may take up to a few minutes to complete** - reduce the number of active points for a faster calculation.

[130]:
```
from special.nested_sampling import nested_spec_sampling

res = nested_spec_sampling(ini_guess, lbda, spec, spec_err, d_st, dlbda_obs=dlbda,␣
↪**instru_params,
                           **nested_params, **model_params, **output_params)
```

```
Fits HDU-0 data successfully loaded. Data shape: (21, 7, 42, 2)
--------------------------------------------------------------------------------
Starting time: 2023-05-08 17:12:50
--------------------------------------------------------------------------------
Prior bounds on parameters:
Teff [2000,4000]
logg [2.5,5.5]
R [0.1,5]
Av [0.0,5]

Using 1000 active points
[ultranest] Resuming from 20543 stored points
```

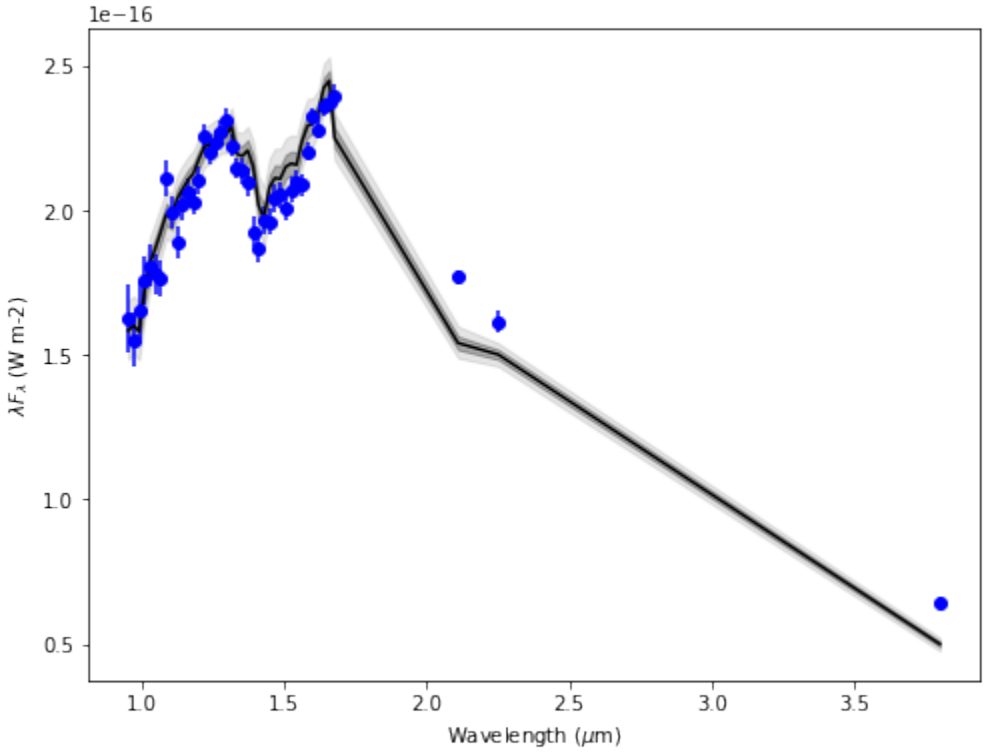
```
VBox(children=(HTML(value=''), GridspecLayout(children=(HTML(value="<div style=
↪'background-color:#6E6BF4;'>&nb...
```

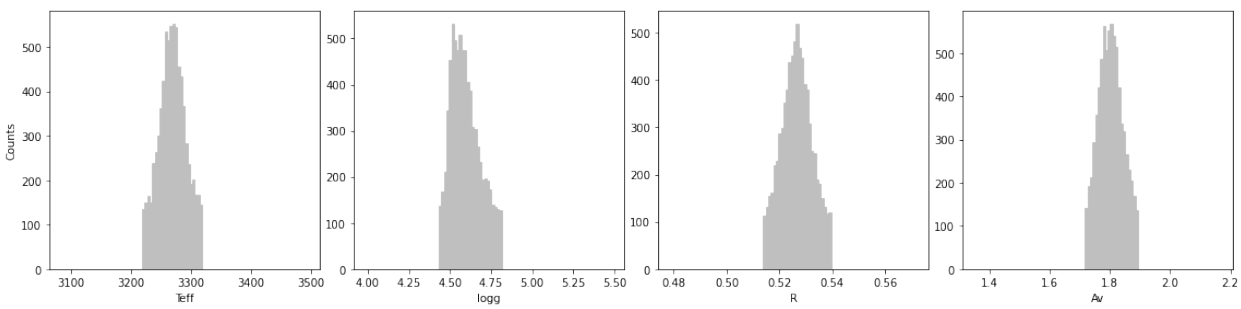
```
[ultranest] Explored until L=-8e+01   [-79.4751..-79.4751]*| it/evals=19500/49465 eff=inf
↪% N=1000    00   0 00 0
[ultranest] Likelihood function evaluations: 49465
[ultranest] Writing samples and results to disk ...
[ultranest] Writing samples and results to disk ... done
[ultranest]   logZ = -94.36 +- 0.08149
[ultranest] Effective samples strategy satisfied (ESS = 5363.6, need >400)
[ultranest] Posterior uncertainty strategy is satisfied (KL: 0.46+-0.05 nat, need <0.50␣
↪nat)
[ultranest] Evidency uncertainty strategy is satisfied (dlogz=0.08, need <0.2)
[ultranest]   logZ error budget: single: 0.11 bs:0.08 tail:0.01 total:0.08 required:<0.20
[ultranest] done iterating.

Total running time:
Running time:  0:00:23.576898
--------------------------------------------------------------------------------
```

Now let's plot the results:

[131]:
```
from special.nested_sampling import show_ultranest_results

del model_params["bounds"] # unused for ultranest results plot
del instru_params["instru_corr"] # unused for ultranest results plot
final_res = show_ultranest_results(res, bins=None, cfd=68.27, save=False, plot=True,
                                   units=units, ndig=ndig, labels_plot=labels_plot,
                                   lbda_obs=lbda, spec_obs=spec, spec_obs_err=spec_err,␣
↪dist=d_st,
                                   **model_params, **instru_params)
```

```
logZ = -94.355 +- 0.125
  single instance: logZ = -94.355 +- 0.114
  bootstrapped   : logZ = -94.361 +- 0.125
  tail           : logZ = +- 0.010
insert order U test : converged: True correlation: inf iterations

    Teff                : 3084   |     |3494      3270 +- 44
    logg                : 4.00   |   |5.38      4.66 +- 0.18
    R                   : 0.480  |    |0.574     0.527 +- 0.011
    Av                  : 1.368  |       |2.172     1.808 +- 0.083


Confidence intervals

Hist bins = 98
percentage for Teff: 69.05748319588042%
percentage for logg: 68.61098073412055%
percentage for R: 68.49686732810508%
percentage for Av: 69.70769181174423%
******* Results for Teff *****


Confidence intervals:
Teff: 3258.5838073052732 [-39.84428271142815,60.81495782270622]
******* Results for logg *****


Confidence intervals:
logg: 4.577720445280006 [-0.1451559643056921,0.2368334154461298]
******* Results for R *****


Confidence intervals:
R: 0.5256796962813602 [-0.011878513577833094,0.013779075750286363]
******* Results for Av *****


Confidence intervals:
Av: 1.829998138709031 [-0.11273860280808967,0.06263255711560567]
```



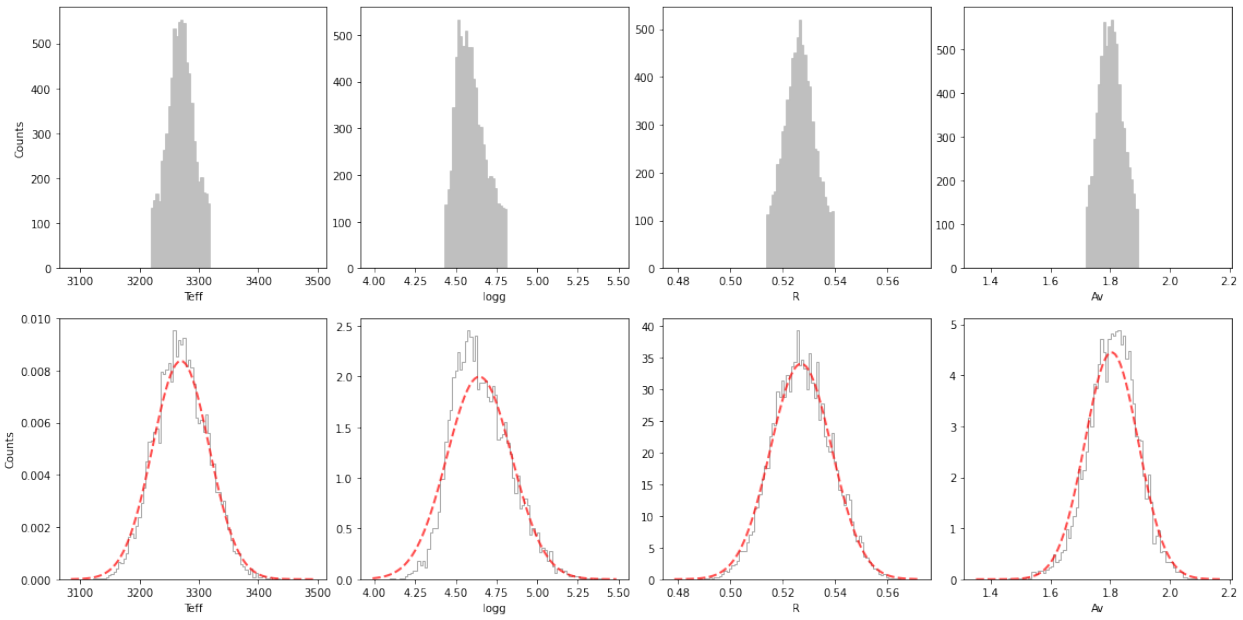



```
percentage for Teff: 69.05748319588042%
percentage for logg: 68.61098073412055%
percentage for R: 68.49686732810508%
percentage for Av: 69.70769181174423%
******* Results for Teff *****


Confidence intervals:
Teff: 3258.5838073052732 [-39.84428271142815,60.81495782270622]
Gaussian fit results:
Teff: 3269.932232046592 +-47.65653344442819
******* Results for logg *****


Confidence intervals:
logg: 4.577720445280006 [-0.1451559643056921,0.2368334154461298]
Gaussian fit results:
logg: 4.644950396124002 +-0.19952209766044599
```

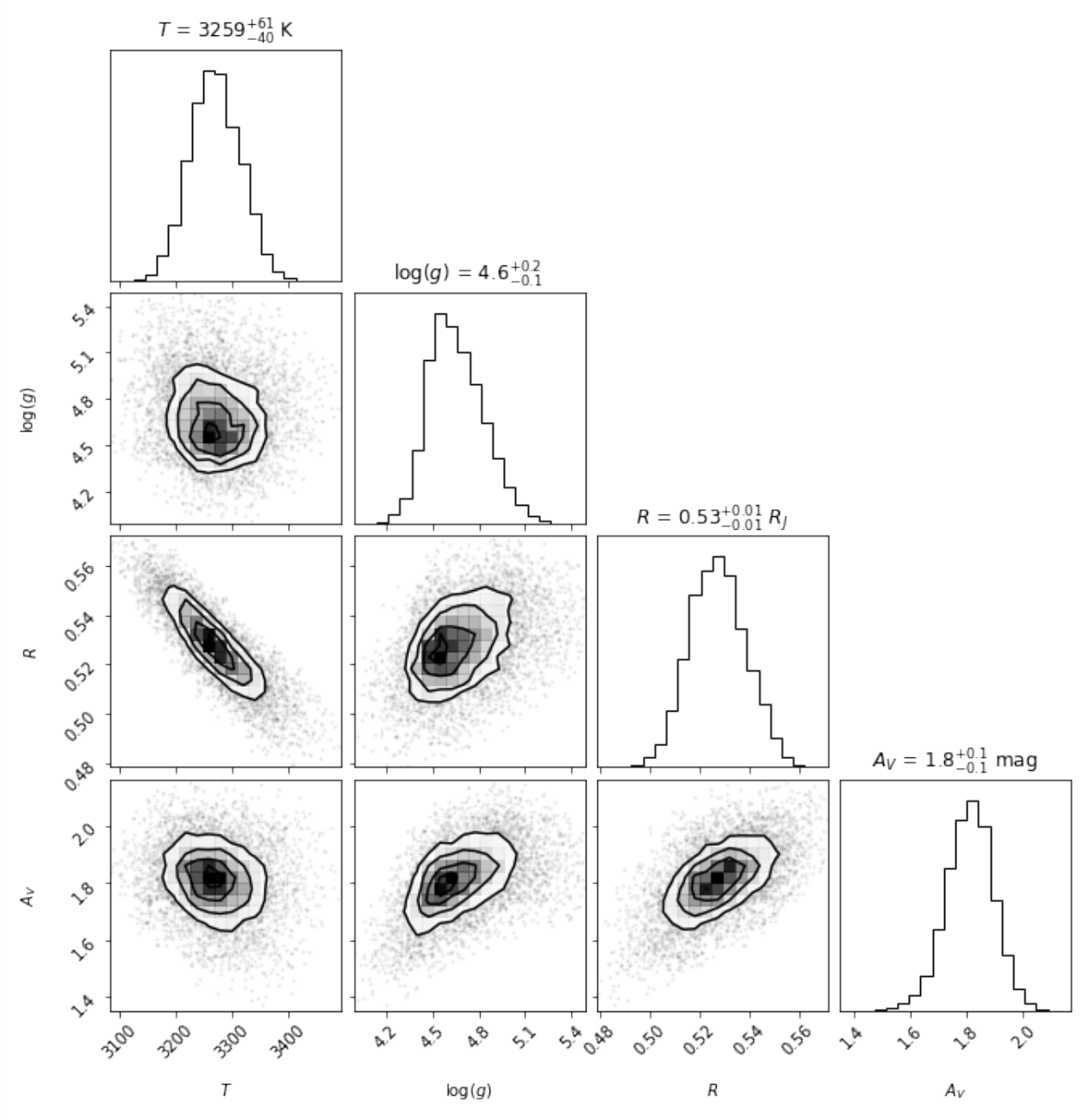
(continues on next page)

(continued from previous page)

```
******* Results for R *****


Confidence intervals:
R: 0.5256796962813602 [-0.011878513577833094,0.013779075750286363]
Gaussian fit results:
R: 0.5270857975433556 +-0.011702451604743036
******* Results for Av *****


Confidence intervals:
Av: 1.829998138709031 [-0.11273860280808967,0.06263255711560567]
Gaussian fit results:
Av: 1.8050380170168783 +-0.08960446777712383
```




```
Teff = 3270 +/- 7
logg = 4.6 +/- 0.4
R = 0.53 +/- 0.11
Av = 1.8 +/- 0.3
```



We see that for the case of the spectrum of CrA-9B/b with a BT-SETTL + extinction model, UltraNest also finds that the highest likelihood area in the parameter space corresponds to an individual ellipsoid.

CHAPTER

ELEVEN

7. BEST-FIT TEMPLATE SPECTRUM

In this section, we will find the best-fit template spectra to our measured spectrum in the Montreal Spectral Library of observed M-L-T type low-mass objects. Since the template spectra from this library typically do not cover wavelengths longward of the K band, we will discard the NACO L' point from the spectrum, just for the sake of this exercise:

[132]:
```
lbda_crop = lbda[:-1]
spec_crop = spec[:-1]
spec_err_crop = spec_err[:-1]
dlbda_crop = dlbda[:-1]
```

Let's define the parameters associated to the instrument(s) which acquired the spectrum (again we discard the NACO measurement):

[133]:
```
instru_res_crop = instru_res[:-1]
instru_idx_crop = instru_idx[:-1]
final_sp_corr_crop = final_sp_corr[:-1,:-1]

instru_params = {'instru_res':instru_res_crop,
                 'instru_idx':instru_idx_crop,
                 'instru_corr':final_sp_corr_crop,
                 'filter_reader':filter_reader}
```

For this exercice, we have downloaded the MSL library following the instructions on the page of J. Gagné: https://jgagneastro.com/the-montreal-spectral-library/

Below, we then set the folder in which the template spectra placed, and their extension ('.fits', for automatic identification of spectra in that folder). We also define the snippet function `tmp_reader` to read the template spectra in the utils.py file, loaded below.

[134]:
```
from utils import tmp_reader
rel_path = '../static/MSL/'
inpath_models = str((par_path / rel_path).resolve())+'/'
tmp_endswith='.fits'
tmp_params = {'tmp_reader':tmp_reader,
              'lib_dir':inpath_models,
              'tmp_endswith':tmp_endswith}
```

FYI it is commented below:

[135]:
```
# def tmp_reader(tmp_name, verbose=False):
#     rel_path = '../static/MSL/'
#     inpath_tmp = str((par_path / rel_path).resolve())+'/'
#     if verbose:
```

(continues on next page)

(continued from previous page)

```
#         print("opening {}".format(tmp_name))
#     spec_model = open_fits(inpath_tmp+tmp_name, verbose=False,
#                            output_verify='ignore')
#     lbda_model = spec_model[0]
#     flux_model = spec_model[1]
#     flux_model_err = spec_model[2]
#     return lbda_model, flux_model, flux_model_err
```

Let's assume we are interested in the top 3 most similar template spectra:

[136]:
```
n_best=3
```

Let's use the Nelder-Mead algorithm to find the optimal flux scaling factor and A_V to be applied to each template. Additional options for the simplex can be set with `simplex_options` (example above) - these will be provided to the `scipy.minimize` function.

[137]:
```
search_mode = 'simplex' #'grid'
simplex_options = {'xatol': 1e-6, 'fatol': 1e-6, 'maxiter': 1000, 'maxfev': 5000}
AV_range = (-3, 7, 0.1) # min, max, step. Note: in simplex mode, min and max values are␣
↪only used to set a chi=np.inf outside of these bounds.

search_opt = {'search_mode':search_mode,
              'simplex_options':simplex_options,
              #'scale_range':(0.2, 5, 0.05),# min, max, step
              'ext_range':AV_range}
```

Alternatively, one can set `search_mode` to `'grid'`, which will simply use a grid search to find the optimal A_V and flux scaling factors to be applied to the template spectra for comparison to the measured spectrum. This is slower but avoids the risk of a failed minimization. In this case, both the `AV_range` and `scale_range` parameters have to be set.

Some template spectra may be corrupted or unusable for this specific spectrum (e.g. wrong wavelength coverage), let's force the algorithm to continue instead of raising an error.

[138]:
```
force_continue=True
```

Some template spectra will have a sufficient wavelength coverage (Y to K), but with missing points (NaN values) in between bands (e.g. between J and H bands, or H and K bands). For a given template spectrum to be considered valid, it is recommeded to set a minimum threshold in terms of number of non-NaN resampled points. The risk of not doing that is to end up with best-fit template spectra with very few - but very well fitting - points minimizing our χ^2_r metric.

Let's set this threshold to cover >80% of our observed spectrum, i.e. 34 points:

[139]:
```
min_npts = 34
```

Now let's find the best-fit templates:

[140]:
```
from special.template_fit import best_fit_tmp

final_res = best_fit_tmp(lbda_crop, spec_crop, spec_err_crop, dlbda_obs=dlbda_crop, n_
↪best=n_best,
                         **tmp_params, **search_opt, **instru_params, force_
↪continue=force_continue,
                         verbosity=1, nproc=1, min_npts=min_npts)
```

```
--------------------------------------------------------------------------------
Starting time: 2023-05-08 17:22:55
--------------------------------------------------------------------------------
424 template spectra will be tested.
****************************************

Unsufficient number of good points (33 < 34). Tmp discarded.
1/424 (2MASSJ0428+1839_NIR_SpeX_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0313-2447_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
2/424 (2MASSJ0313-2447_NIR_Flamingos-2_Gagne2015c.fits) FAILED
3/424: SIMPJ2215+2110_NIR_SpeX_Robert2016, chi_r^2 = 43.1, ndof=39
Unsufficient number of good points (33 < 34). Tmp discarded.
4/424 (2MASSJ2353-1844A_NIR_SpeX_Gagne2015c.fits) FAILED
Wavelength range of template SIMPJ1150+0520_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
5/424 (SIMPJ1150+0520_NIR_NIRI_Robert2016.fits) FAILED
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template SIMPJ2238+2304_NIR_SpeX_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
6/424 (SIMPJ2238+2304_NIR_SpeX_Robert2016.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2353A_merge'       /                                             ␣
↪[astropy.io.fits.card]
7/424: 2MASSJ1045-2819_NIR_SpeX_Gagne2015c, chi_r^2 = 5.4, ndof=33
8/424: SIMPJ1501-1831_NIR_SpeX_Robert2016, chi_r^2 = 10.5, ndof=39
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ2251-6811_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.47)mu?
9/424 (2MASSJ2251-6811_NIR_Flamingos-2_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2251-6811_ALLCOMB_merge' /                                       ␣
↪[astropy.io.fits.card]
10/424: SIMPJ0429+0607_NIR_GNIRS_Robert2016, chi_r^2 = 34.6, ndof=39
11/424: SIMPJ1058+1339_NIR_SIMON_Robert2016, chi_r^2 = 22.4, ndof=39
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ0752-7947_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.47)mu?
12/424 (2MASSJ0752-7947_NIR_Flamingos-2_Gagne2015c.fits) FAILED
13/424: 2MASSJ0443+0002_NIR_SpeX_Gagne2015c, chi_r^2 = 14.7, ndof=39
Wavelength range of template SIMPJ2148+2239_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
14/424 (SIMPJ2148+2239_NIR_NIRI_Robert2016.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2202-5605_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
15/424 (2MASSJ2202-5605_NIR_Flamingos-2_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2202-5605_ALLCOMB_merge' /                                              ␣
↪[astropy.io.fits.card]
```

```
16/424: SIMPJ0951-1343_NIR_SIMON_Robert2016, chi_r^2 = 33.6, ndof=39
17/424: 2MASSJ1119-3917_NIR_SpeX_Gagne2015c, chi_r^2 = 2.6, ndof=33
18/424: SIMPJ1627+0836_NIR_SpeX_Robert2016, chi_r^2 = 11.3, ndof=39
19/424: 2MASSJ0344+0716B_NIR_SpeX_Gagne2015c, chi_r^2 = 3.3, ndof=33
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template SIMPJ1510-1147_NIR_SpeX_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.43)mu?
20/424 (SIMPJ1510-1147_NIR_SpeX_Robert2016.fits) FAILED
21/424: SIMPJ1415+2635_NIR_SpeX_Robert2016, chi_r^2 = 14.5, ndof=39
Wavelength range of template SIMPJ1613-0747_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
22/424 (SIMPJ1613-0747_NIR_NIRI_Robert2016.fits) FAILED
Unsufficient number of good points (30 < 34). Tmp discarded.
23/424 (2MASSJ2240+0532_NIR_SpeX_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2240+0532_merge'   /                                                    ␣
↪[astropy.io.fits.card]
```

```
24/424: SIMPJ2322-1407_NIR_SpeX_Robert2016, chi_r^2 = 26.6, ndof=39
Wavelength range of template SIMPJ1225-1013_NIR_GNIRS_Robert2016.fits (1.00, 2.52)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
25/424 (SIMPJ1225-1013_NIR_GNIRS_Robert2016.fits) FAILED
No indices match the constraint (ceil w.r.t 2.36)
Issue with resampling of template SIMPJ2318-1301_NIR_SIMON_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.80, 2.27)mu?
26/424 (SIMPJ2318-1301_NIR_SIMON_Robert2016.fits) FAILED
27/424: SIMPJ1217+0708_NIR_GNIRS_Robert2016, chi_r^2 = 14.4, ndof=35
28/424: SIMPJ0858+2710_NIR_GNIRS_Robert2016, chi_r^2 = 115.6, ndof=35
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0120-5200_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
29/424 (2MASSJ0120-5200_NIR_Flamingos-2_Gagne2015c.fits) FAILED
30/424: SIMPJ0150+3827_NIR_SpeX_Robert2016, chi_r^2 = 30.7, ndof=39
Unsufficient number of good points (31 < 34). Tmp discarded.
31/424 (2MASSJ0103-2805B_NIR_SpeX_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0041-5621_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
32/424 (2MASSJ0041-5621_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Wavelength range of template SIMPJ1756+2815_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
33/424 (SIMPJ1756+2815_NIR_NIRI_Robert2016.fits) FAILED
34/424: SIMPJ1031+3349_NIR_SpeX_Robert2016, chi_r^2 = 15.3, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2041-3506_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
35/424 (2MASSJ2041-3506_NIR_Flamingos-2_Gagne2015c.fits) FAILED
```

(continues on next page)

(continued from previous page)

```
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2219-6828_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
36/424 (2MASSJ2219-6828_NIR_Flamingos-2_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2041-3506_ALLCOMB_merge' /                                       ␣
↪[astropy.io.fits.card]
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J22191486-6828018_ALLCOMB_merge' /                                ␣
↪[astropy.io.fits.card]
```

```
37/424: SIMPJ1211+0406_NIR_SIMON_Robert2016, chi_r^2 = 18.2, ndof=39
38/424: SIMPJ0148+3712_NIR_SpeX_Robert2016, chi_r^2 = 17.2, ndof=39
39/424: SIMPJ1308+0818_NIR_SpeX_Robert2016, chi_r^2 = 9.1, ndof=39
40/424: SIMPJ0851+1043_NIR_GNIRS_Robert2016, chi_r^2 = 201.1, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0226-5327_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
41/424 (2MASSJ0226-5327_NIR_Flamingos-2_Gagne2015c.fits) FAILED
42/424: 2MASSJ0335+2342_NIR_SpeX_Gagne2015c, chi_r^2 = 7.8, ndof=39
43/424: SIMPJ2150-0412_NIR_SIMON_Robert2016, chi_r^2 = 24.8, ndof=39
Wavelength range of template SIMPJ1755+3618_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
44/424 (SIMPJ1755+3618_NIR_NIRI_Robert2016.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0541-0217_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
45/424 (2MASSJ0541-0217_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ0042+1142_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
46/424 (2MASSJ0042+1142_NIR_SpeX_Gagne2015c.fits) FAILED
47/424: SIMPJ0417-1838_NIR_SpeX_Robert2016, chi_r^2 = 12.3, ndof=39
48/424: SIMPJ0947+0617_NIR_SIMON_Robert2016, chi_r^2 = 8.2, ndof=39
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2244-3045_merge'   /                                             ␣
↪[astropy.io.fits.card]
```

```
49/424: 2MASSJ2244-3045_NIR_SpeX_Gagne2015c, chi_r^2 = 13.0, ndof=35
Wavelength range of template SIMPJ1414+0107_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
50/424 (SIMPJ1414+0107_NIR_NIRI_Robert2016.fits) FAILED
51/424: 2MASSJ0758+15301A_NIR_SpeX_Gagne2015c, chi_r^2 = 3.1, ndof=32
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0038-6403_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
52/424 (2MASSJ0038-6403_NIR_Flamingos-2_Gagne2015c.fits) FAILED
53/424: SIMPJ1058+1339_NIR_GNIRS_Robert2016, chi_r^2 = 36.0, ndof=39
54/424: SIMPJ0429+0607_NIR_SIMON_Robert2016, chi_r^2 = 24.5, ndof=39
```

(continues on next page)

(continued from previous page)

```
55/424: 2MASSJ1411-2119_NIR_SpeX_Gagne2015c, chi_r^2 = 9.5, ndof=39
56/424: 2MASSJ0856-1342_NIR_SpeX_Gagne2015c, chi_r^2 = 10.1, ndof=39
57/424: 2MASSJ1207-3900_NIR_SpeX_Gagne2015c, chi_r^2 = 19.4, ndof=39
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template SIMPJ0418+1121_NIR_SpeX_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.43)mu?
58/424 (SIMPJ0418+1121_NIR_SpeX_Robert2016.fits) FAILED
59/424: 2MASSJ0337-3709_NIR_SpeX_Gagne2015c, chi_r^2 = 3.1, ndof=33
60/424: SIMPJ2155+2345_NIR_SpeX_Robert2016, chi_r^2 = 15.3, ndof=39
61/424: SIMPJ2249-1628_NIR_SpeX_Robert2016, chi_r^2 = 25.4, ndof=39
62/424: SIMPJ1411+2948_NIR_SpeX_Robert2016, chi_r^2 = 39.4, ndof=39
63/424: SIMPJ1453-2744_NIR_GNIRS_Robert2016, chi_r^2 = 94.9, ndof=39
64/424: 2MASSJ0046+0715_NIR_SpeX_Gagne2015c, chi_r^2 = 13.5, ndof=32
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ2335-3401_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.47)mu?
65/424 (2MASSJ2335-3401_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0309-3014_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
66/424 (2MASSJ0309-3014_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0019-6226_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
67/424 (2MASSJ0019-6226_NIR_Flamingos-2_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2335-3401_ALLCOMB_merge' /                                       ␣
↪[astropy.io.fits.card]
68/424: 2MASSJ0404+2616B_NIR_SpeX_Gagne2015c, chi_r^2 = 8.1, ndof=32
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ1547-1626B_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
69/424 (2MASSJ1547-1626B_NIR_SpeX_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ1227-0636_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
70/424 (2MASSJ1227-0636_NIR_Flamingos-2_Gagne2015c.fits) FAILED
71/424: 2MASSJ1948+5944B_NIR_SpeX_Gagne2015c, chi_r^2 = 3.5, ndof=35
Unsufficient number of good points (33 < 34). Tmp discarded.
72/424 (2MASSJ2011-5048_NIR_FIRE_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0259-4232_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
73/424 (2MASSJ0259-4232_NIR_Flamingos-2_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'W2011-5048_tc.fits' /                                             ␣
↪[astropy.io.fits.card]
74/424: 2MASSJ1153-3015_NIR_SpeX_Gagne2015c, chi_r^2 = 2.7, ndof=33
```

(continues on next page)

(continued from previous page)

```
75/424: SIMPJ2303+3150_NIR_SpeX_Robert2016, chi_r^2 = 70.2, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ1156-4043_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
76/424 (2MASSJ1156-4043_NIR_Flamingos-2_Gagne2015c.fits) FAILED
77/424: SIMPJ1211+0406_NIR_GNIRS_Robert2016, chi_r^2 = 107.9, ndof=36
78/424: 2MASSJ1035-2058_NIR_SpeX_Gagne2015c, chi_r^2 = 3.3, ndof=33
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ0314+1603_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
79/424 (2MASSJ0314+1603_NIR_SpeX_Gagne2015c.fits) FAILED
80/424: SIMPJ1317+0448_NIR_SpeX_Robert2016, chi_r^2 = 14.4, ndof=39
81/424: 2MASSJ0001+1535_NIR_FIRE_Gagne2015c, chi_r^2 = 20.9, ndof=39
82/424: SIMPJ1343-1216_NIR_SIMON_Robert2016, chi_r^2 = 27.4, ndof=39
2MASSJ0039+1330B_NIR_SpeX_Gagne2015c.fits could not be opened. Corrupt file?
83/424 (2MASSJ0039+1330B_NIR_SpeX_Gagne2015c.fits) FAILED
84/424: 2MASSJ1013-1706A_NIR_SpeX_Gagne2015c, chi_r^2 = 3.0, ndof=39
85/424: SIMPJ0417+1634_NIR_SIMON_Robert2016, chi_r^2 = 14.4, ndof=39
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template SIMPJ0422+1033_NIR_SpeX_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.43)mu?
86/424 (SIMPJ0422+1033_NIR_SpeX_Robert2016.fits) FAILED
Unsufficient number of good points (31 < 34). Tmp discarded.
87/424 (2MASSJ1846+5246B_NIR_SpeX_Gagne2015c.fits) FAILED
88/424: 2MASSJ0602-1413_NIR_SpeX_Gagne2015c, chi_r^2 = 12.2, ndof=39
89/424: 2MASSJ1106-3715_NIR_SpeX_Gagne2015c, chi_r^2 = 10.4, ndof=33
90/424: 2MASSJ0527+0007B_NIR_SpeX_Gagne2015c, chi_r^2 = 3.1, ndof=33
91/424: 2MASSJ1148-2836_NIR_Flamingos-2_Gagne2015c, chi_r^2 = 15.1, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2244-6650_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
92/424 (2MASSJ2244-6650_NIR_Flamingos-2_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2244-6650_ALLCOMB_merge' /                                              ␣
↪[astropy.io.fits.card]
93/424: SIMPJ1039-1904_NIR_SpeX_Robert2016, chi_r^2 = 13.0, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ1133-7807_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
94/424 (2MASSJ1133-7807_NIR_Flamingos-2_Gagne2015c.fits) FAILED
95/424: 2MASSJ1207-3900_NIR_SpeX_Gagne2014b, chi_r^2 = 19.4, ndof=39
96/424: SIMPJ0135+0205_NIR_SIMON_Robert2016, chi_r^2 = 23.4, ndof=39
97/424: SIMPJ0956-1447_NIR_SpeX_Robert2016, chi_r^2 = 37.4, ndof=39
Unsufficient number of good points (33 < 34). Tmp discarded.
98/424 (2MASSJ2335-1908_NIR_SpeX_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0819-0450_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
99/424 (2MASSJ0819-0450_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Wavelength range of template SIMPJ1158+0435_NIR_SpeX_Robert2016.fits (1.13, 2.43)mu too␣
```

(continues on next page)

(continued from previous page)

```
↪short compared to that of observed spectrum (0.95, 2.25)mu
100/424 (SIMPJ1158+0435_NIR_SpeX_Robert2016.fits) FAILED
2MASSJ0002+0408A_NIR_SpeX_Gagne2015c.fits could not be opened. Corrupt file?
101/424 (2MASSJ0002+0408A_NIR_SpeX_Gagne2015c.fits) FAILED
Unsufficient number of good points (33 < 34). Tmp discarded.
102/424 (2MASSJ2343-3646_NIR_FIRE_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2335-1908_merge'   /                                              ␣
↪[astropy.io.fits.card]
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'W2343-3646_tc.fits' /                                              ␣
↪[astropy.io.fits.card]
```

```
103/424: SIMPJ0006-1319_NIR_GNIRS_Robert2016, chi_r^2 = 135.5, ndof=35
104/424: SIMPJ0532-3253_NIR_GNIRS_Robert2016, chi_r^2 = 216.5, ndof=38
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0148-5201_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
105/424 (2MASSJ0148-5201_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ2028-5637_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.47)mu?
106/424 (2MASSJ2028-5637_NIR_Flamingos-2_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2028-5637_ALLCOMB_merge' /                                        ␣
↪[astropy.io.fits.card]
```

```
107/424: 2MASSJ0758+15300_NIR_SpeX_Gagne2015c, chi_r^2 = 2.9, ndof=32
108/424: SIMPJ1324+1906_NIR_GNIRS_Robert2016, chi_r^2 = 88.3, ndof=39
Wavelength range of template 2MASSJ2235-5906_NIR_Flamingos-2_Gagne2015c.fits (0.89, 1.
↪89)mu too short compared to that of observed spectrum (0.95, 2.25)mu
109/424 (2MASSJ2235-5906_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0129-0823_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
110/424 (2MASSJ0129-0823_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0819-7401_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
111/424 (2MASSJ0819-7401_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0537-0623_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
112/424 (2MASSJ0537-0623_NIR_Flamingos-2_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2235-5906_BEAMS_AB_140710_386_387_388_389_comb.fits' /            ␣
↪[astropy.io.fits.card]
```

```
113/424: SIMPJ1307-0558_NIR_GNIRS_Robert2016, chi_r^2 = 262.0, ndof=36
114/424: 2MASSJ1207-3932_NIR_SpeX_Gagne2015c, chi_r^2 = 10.2, ndof=33
Wavelength range of template SIMPJ1118-0856_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
115/424 (SIMPJ1118-0856_NIR_NIRI_Robert2016.fits) FAILED
116/424: SIMPJ1629+0335_NIR_SIMON_Robert2016, chi_r^2 = 49.5, ndof=39
117/424: SIMPJ0357+1529_NIR_GNIRS_Robert2016, chi_r^2 = 20.1, ndof=39
118/424: SIMPJ0400-1322_NIR_GNIRS_Robert2016, chi_r^2 = 124.5, ndof=38
Wavelength range of template 2MASSJ1207-3900_OPT_MAGE_Gagne2014b.fits (38240.71, 43051.
↪20)mu too short compared to that of observed spectrum (0.95, 2.25)mu
119/424 (2MASSJ1207-3900_OPT_MAGE_Gagne2014b.fits) FAILED
120/424: SIMPJ1627+0546_NIR_SpeX_Robert2016, chi_r^2 = 8.2, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2000-7523_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
121/424 (2MASSJ2000-7523_NIR_Flamingos-2_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J20004841-7523070_ALLCOMB_merge' /                                 ␣
↪[astropy.io.fits.card]
WARNING: non-ASCII characters are present in the FITS file header and have been replaced␣
↪by "?" characters [astropy.io.fits.util]
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'spc0070.a.fits'                                                    ␣
↪[astropy.io.fits.card]
```

```
122/424: 2MASSJ2154-1055_NIR_SpeX_Gagne2015c, chi_r^2 = 22.7, ndof=39
123/424: 2MASSJ0536-1920_NIR_SpeX_Gagne2015c, chi_r^2 = 39.8, ndof=32
124/424: SIMPJ0422+1033_NIR_SIMON_Robert2016, chi_r^2 = 9.9, ndof=39
Wavelength range of template SIMPJ1333-0215_NIR_SpeX_Robert2016.fits (1.13, 2.43)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
125/424 (SIMPJ1333-0215_NIR_SpeX_Robert2016.fits) FAILED
126/424: SIMPJ1452+1114_NIR_SpeX_Robert2016, chi_r^2 = 15.4, ndof=39
Unsufficient number of good points (31 < 34). Tmp discarded.
127/424 (2MASSJ2310-0748_NIR_SpeX_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0649-3823_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
128/424 (2MASSJ0649-3823_NIR_Flamingos-2_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2310-0748_merge'   /                                              ␣
↪[astropy.io.fits.card]
```

```
129/424: SIMPJ1613-0747_NIR_GNIRS_Robert2016, chi_r^2 = 47.5, ndof=38
130/424: SIMPJ0552+0210_NIR_GNIRS_Robert2016, chi_r^2 = 203.1, ndof=36
131/424: SIMPJ0417-1345_NIR_SpeX_Robert2016, chi_r^2 = 10.5, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0018-6703_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
132/424 (2MASSJ0018-6703_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
```

(continues on next page)

(continued from previous page)

```
Issue with resampling of template 2MASSJ0300-5459_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
133/424 (2MASSJ0300-5459_NIR_Flamingos-2_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'W2315-4747_tc.fits' /                                             ␣
↪[astropy.io.fits.card]
```

```
134/424: 2MASSJ2315-4747_NIR_FIRE_Gagne2015c, chi_r^2 = 13.6, ndof=33
135/424: SIMPJ0316+2650_NIR_SpeX_Robert2016, chi_r^2 = 65.4, ndof=39
Unsufficient number of good points (30 < 34). Tmp discarded.
136/424 (2MASSJ1621-2346_NIR_SpeX_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0320-3313_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
137/424 (2MASSJ0320-3313_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ1101-7735_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
138/424 (2MASSJ1101-7735_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Unsufficient number of good points (33 < 34). Tmp discarded.
139/424 (2MASSJ2048-5127_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Wavelength range of template SIMPJ1004-1127_NIR_GNIRS_Robert2016.fits (1.00, 2.52)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
140/424 (SIMPJ1004-1127_NIR_GNIRS_Robert2016.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'W2048-5127_tc.fits' /                                             ␣
↪[astropy.io.fits.card]
```

```
141/424: 2MASSJ0339-2434_NIR_SpeX_Gagne2015c, chi_r^2 = 3.8, ndof=38
Unsufficient number of good points (33 < 34). Tmp discarded.
142/424 (2MASSJ1529+6312_NIR_SpeX_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0355-1032_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
143/424 (2MASSJ0355-1032_NIR_Flamingos-2_Gagne2015c.fits) FAILED
144/424: SIMPJ0414+1529_NIR_SpeX_Robert2016, chi_r^2 = 7.7, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ1257-4111_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
145/424 (2MASSJ1257-4111_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ2048-3255_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
146/424 (2MASSJ2048-3255_NIR_SpeX_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2048-3255_merge'   /                                             ␣
↪[astropy.io.fits.card]
```

```
147/424: SIMPJ1147+2153_NIR_SpeX_Robert2016, chi_r^2 = 12.5, ndof=39
SIMPJ2332-1249_NIR_SpeX_Robert2016.fits could not be opened. Corrupt file?
148/424 (SIMPJ2332-1249_NIR_SpeX_Robert2016.fits) FAILED
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ0635-6234_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.47)mu?
149/424 (2MASSJ0635-6234_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (ceil w.r.t 2.36)
Issue with resampling of template SIMPJ0008+0806_NIR_SIMON_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.80, 2.35)mu?
150/424 (SIMPJ0008+0806_NIR_SIMON_Robert2016.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2327+3858_merge'   /                                                    ␣
↪[astropy.io.fits.card]
151/424: 2MASSJ2327+3858_NIR_SpeX_Gagne2015c, chi_r^2 = 4.8, ndof=32
152/424: 2MASSJ1358-0046_NIR_SpeX_Gagne2015c, chi_r^2 = 4.7, ndof=39
153/424: SIMPJ2327+1517_NIR_SpeX_Robert2016, chi_r^2 = 19.8, ndof=39
154/424: SIMPJ0049+0440_NIR_GNIRS_Robert2016, chi_r^2 = 30.8, ndof=39
155/424: 2MASSJ0825-0029_NIR_FIRE_Gagne2015c, chi_r^2 = 9.2, ndof=32
156/424: SIMPJ2154-1055_NIR_SpeX_Gagne2014c, chi_r^2 = 22.7, ndof=39
157/424: SIMPJ0006-1319_NIR_SIMON_Robert2016, chi_r^2 = 27.0, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0244-3548_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
158/424 (2MASSJ0244-3548_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Unsufficient number of good points (29 < 34). Tmp discarded.
159/424 (2MASSJ0342-2904_NIR_FIRE_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ0311+0106_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.46)mu?
160/424 (2MASSJ0311+0106_NIR_Flamingos-2_Gagne2015c.fits) FAILED
161/424: SIMPJ1628+0517_NIR_SIMON_Robert2016, chi_r^2 = 35.5, ndof=39
162/424: SIMPJ1324+1906_NIR_SIMON_Robert2016, chi_r^2 = 77.3, ndof=39
163/424: SIMPJ0921-1534_NIR_GNIRS_Robert2016, chi_r^2 = 105.4, ndof=35
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ1706-1314_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
164/424 (2MASSJ1706-1314_NIR_SpeX_Gagne2015c.fits) FAILED
165/424: SIMPJ1004-1318_NIR_GNIRS_Robert2016, chi_r^2 = 25.8, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0322-7940_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
166/424 (2MASSJ0322-7940_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0051-6227_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
167/424 (2MASSJ0051-6227_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Unsufficient number of good points (31 < 34). Tmp discarded.
168/424 (2MASSJ1642-1942_NIR_FIRE_Gagne2015c.fits) FAILED
169/424: 2MASSJ0953-1014_NIR_SpeX_Gagne2015c, chi_r^2 = 14.0, ndof=39
170/424: SIMPJ1629+0335_NIR_GNIRS_Robert2016, chi_r^2 = 55.5, ndof=37
```

(continues on next page)

(continued from previous page)

```
Wavelength range of template SIMPJ2132-1452_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
171/424 (SIMPJ2132-1452_NIR_NIRI_Robert2016.fits) FAILED
Unsufficient number of good points (32 < 34). Tmp discarded.
172/424 (2MASSJ0825+0340_NIR_SpeX_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2149-6413_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
173/424 (2MASSJ2149-6413_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Wavelength range of template SIMPJ1039-2829_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
174/424 (SIMPJ1039-2829_NIR_NIRI_Robert2016.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2149-6413_ALLCOMB_merge' /                                              ␣
↪[astropy.io.fits.card]
175/424: 2MASSJ0820-7514_NIR_FIRE_Gagne2015c, chi_r^2 = 13.7, ndof=33
176/424: SIMPJ2309+1003_NIR_GNIRS_Robert2016, chi_r^2 = 83.9, ndof=39
177/424: 2MASSJ0805+2505B_NIR_SpeX_Gagne2015c, chi_r^2 = 4.4, ndof=39
178/424: SIMPJ1613-0747_NIR_SIMON_Robert2016, chi_r^2 = 14.0, ndof=39
179/424: 2MASSJ0951+3558_NIR_SpeX_Gagne2015c, chi_r^2 = 3.9, ndof=39
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ2325-0259_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
180/424 (2MASSJ2325-0259_NIR_SpeX_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ0407+1546_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
181/424 (2MASSJ0407+1546_NIR_SpeX_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2325-0259'         /                                                    ␣
↪[astropy.io.fits.card]
182/424: SIMPJ2254+1640_NIR_SpeX_Robert2016, chi_r^2 = 14.4, ndof=39
183/424: 2MASSJ1623-2353_NIR_SpeX_Gagne2015c, chi_r^2 = 14.9, ndof=39
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template GUPscb_NIR_GNIRS_Naud2014.fits. Does the wavelength␣
↪range extend far enough (0.94, 2.35)mu?
184/424 (GUPscb_NIR_GNIRS_Naud2014.fits) FAILED
Wavelength range of template SIMPJ1132-3809_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
185/424 (SIMPJ1132-3809_NIR_NIRI_Robert2016.fits) FAILED
186/424: 2MASSJ0526-1824_NIR_SpeX_Gagne2015c, chi_r^2 = 5.0, ndof=34
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ1204-2806_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
187/424 (2MASSJ1204-2806_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0540-0923_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
```

(continues on next page)

(continued from previous page)

```
188/424 (2MASSJ0540-0923_NIR_Flamingos-2_Gagne2015c.fits) FAILED
189/424: 2MASSJ0507+1430A_NIR_SpeX_Gagne2015c, chi_r^2 = 3.6, ndof=39
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ2112-8128_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.47)mu?
190/424 (2MASSJ2112-8128_NIR_Flamingos-2_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2112-8128_ALLCOMB_merge' /                                       ␣
↪[astropy.io.fits.card]
```

```
191/424: SIMPJ0013+0841_NIR_SIMON_Robert2016, chi_r^2 = 20.7, ndof=39
Unsufficient number of good points (33 < 34). Tmp discarded.
192/424 (2MASSJ1939-5216_NIR_FIRE_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2148-4736_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
193/424 (2MASSJ2148-4736_NIR_Flamingos-2_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'W1939-5216_tc.fits' /                                             ␣
↪[astropy.io.fits.card]
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2148-4736_ALLCOMB_merge' /                                       ␣
↪[astropy.io.fits.card]
```

```
194/424: SIMPJ1117+1857_NIR_GNIRS_Robert2016, chi_r^2 = 286.3, ndof=36
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ0355+1133_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
195/424 (2MASSJ0355+1133_NIR_SpeX_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2005-6258_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
196/424 (2MASSJ2005-6258_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template SIMPJ0211-1427_NIR_SpeX_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
197/424 (SIMPJ0211-1427_NIR_SpeX_Robert2016.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2005-6258_ALLCOMB_merge' /                                       ␣
↪[astropy.io.fits.card]
```

```
198/424: SIMPJ1041-0429_NIR_GNIRS_Robert2016, chi_r^2 = 21.0, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0804-6346_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
199/424 (2MASSJ0804-6346_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Wavelength range of template SIMPJ1343-1216_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
```

(continues on next page)

(continued from previous page)

```
200/424 (SIMPJ1343-1216_NIR_NIRI_Robert2016.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ1903-3723_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
201/424 (2MASSJ1903-3723_NIR_Flamingos-2_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J1903-3723_ALLCOMB_merge' /                                              ␣
↪[astropy.io.fits.card]
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J1948+5944A_merge'  /                                                    ␣
↪[astropy.io.fits.card]
202/424: 2MASSJ1948+5944A_NIR_SpeX_Gagne2015c, chi_r^2 = 3.1, ndof=32
Wavelength range of template 2MASSJ1846-5706_NIR_Flamingos-2_Gagne2015c.fits (0.90, 1.
↪90)mu too short compared to that of observed spectrum (0.95, 2.25)mu
203/424 (2MASSJ1846-5706_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ0453-1751_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
204/424 (2MASSJ0453-1751_NIR_SpeX_Gagne2015c.fits) FAILED
Unsufficient number of good points (33 < 34). Tmp discarded.
205/424 (2MASSJ2048-5127_NIR_FIRE_Gagne2015c.fits) FAILED
206/424: SIMPJ1537-0800_NIR_GNIRS_Robert2016, chi_r^2 = 169.0, ndof=36
Wavelength range of template SIMPJ2322-1407_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
207/424 (SIMPJ2322-1407_NIR_NIRI_Robert2016.fits) FAILED
208/424: SIMPJ2132-1452_NIR_SIMON_Robert2016, chi_r^2 = 102.1, ndof=39
209/424: 2MASSJ0404+2616A_NIR_SpeX_Gagne2015c, chi_r^2 = 6.8, ndof=33
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ1547-1626A_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
210/424 (2MASSJ1547-1626A_NIR_SpeX_Gagne2015c.fits) FAILED
211/424: SIMPJ0425-1900_NIR_GNIRS_Robert2016, chi_r^2 = 37.9, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ1249-2035_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
212/424 (2MASSJ1249-2035_NIR_Flamingos-2_Gagne2015c.fits) FAILED
213/424: SIMPJ0120+1518_NIR_SpeX_Robert2016, chi_r^2 = 11.2, ndof=39
214/424: SIMPJ2239+1617_NIR_SIMON_Robert2016, chi_r^2 = 61.1, ndof=39
215/424: SIMPJ1708+2606_NIR_SpeX_Robert2016, chi_r^2 = 24.5, ndof=39
216/424: SIMPJ1036+0306_NIR_SIMON_Robert2016, chi_r^2 = 23.2, ndof=39
217/424: SIMPJ0421-1133_NIR_GNIRS_Robert2016, chi_r^2 = 189.6, ndof=35
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ1221+0257_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
218/424 (2MASSJ1221+0257_NIR_Flamingos-2_Gagne2015c.fits) FAILED
219/424: SIMPJ1014+1900_NIR_GNIRS_Robert2016, chi_r^2 = 52.9, ndof=36
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0910-7552_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
```

(continues on next page)

(continued from previous page)

```
220/424 (2MASSJ0910-7552_NIR_Flamingos-2_Gagne2015c.fits) FAILED
221/424: 2MASSJ0520+0511_NIR_SpeX_Gagne2015c, chi_r^2 = 6.9, ndof=32
222/424: SIMPJ0936+0528_NIR_GNIRS_Robert2016, chi_r^2 = 224.4, ndof=38
223/424: SIMPJ1052+1722_NIR_SIMON_Robert2016, chi_r^2 = 14.6, ndof=39
224/424: SIMPJ2117-0611_NIR_SIMON_Robert2016, chi_r^2 = 19.9, ndof=39
No indices match the constraint (floor w.r.t 0.84)
Issue with resampling of template 2MASSJ1625-2358_NIR_GNIRS_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.84, 2.53)mu?
225/424 (2MASSJ1625-2358_NIR_GNIRS_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_F-TLM= '2013-07-18T21:19:33' / Time of last modification                  ␣
↪[astropy.io.fits.card]
```

```
226/424: SIMPJ0251-0818_NIR_SIMON_Robert2016, chi_r^2 = 17.0, ndof=39
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J1846+5246_A_merge' /                                             ␣
↪[astropy.io.fits.card]
```

```
227/424: 2MASSJ1846+5246A_NIR_SpeX_Gagne2015c, chi_r^2 = 8.0, ndof=32
2MASSJ0039+1330A_NIR_SpeX_Gagne2015c.fits could not be opened. Corrupt file?
228/424 (2MASSJ0039+1330A_NIR_SpeX_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ1257-3635_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.47)mu?
229/424 (2MASSJ1257-3635_NIR_Flamingos-2_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J12574463-3635431_ALLCOMB_merge' /                                ␣
↪[astropy.io.fits.card]
```

```
230/424: SIMPJ1150+0520_NIR_SpeX_Robert2016, chi_r^2 = 20.2, ndof=39
231/424: 2MASSJ1013-1706B_NIR_SpeX_Gagne2015c, chi_r^2 = 3.1, ndof=39
232/424: SIMPJ1141+1941_NIR_GNIRS_Robert2016, chi_r^2 = 22.8, ndof=35
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template SIMPJ1003-1441_NIR_SpeX_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.43)mu?
233/424 (SIMPJ1003-1441_NIR_SpeX_Robert2016.fits) FAILED
234/424: SIMPJ1631-1922_NIR_SpeX_Robert2016, chi_r^2 = 3.5, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2235-3844_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
235/424 (2MASSJ2235-3844_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ0326-0617_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
236/424 (2MASSJ0326-0617_NIR_SpeX_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2235-3844_ALLCOMB_merge' /                                       ␣
↪[astropy.io.fits.card]
```

```
237/424: 2MASSJ0518-3101_NIR_SpeX_Gagne2015c, chi_r^2 = 7.2, ndof=37
238/424: SIMPJ1422+0827_NIR_SIMON_Robert2016, chi_r^2 = 31.3, ndof=39
2MASSJ2331-0406_NIR_Flamingos-2_Gagne2015c.fits could not be opened. Corrupt file?
239/424 (2MASSJ2331-0406_NIR_Flamingos-2_Gagne2015c.fits) FAILED
WARNING: VerifyWarning: Error validating header for HDU #0 (note: Astropy uses zero-
↪based indexing).
    Header size is not multiple of 2880: 11366
There may be extra bytes after the last HDU or the file is corrupted. [astropy.io.fits.
↪hdu.hdulist]
240/424: SIMPJ1339+1523_NIR_SIMON_Robert2016, chi_r^2 = 13.0, ndof=39
241/424: 2MASSJ1226-3316_NIR_SpeX_Gagne2015c, chi_r^2 = 3.4, ndof=32
242/424: SIMPJ1615+1340_NIR_SIMON_Robert2016, chi_r^2 = 135.2, ndof=39
243/424: 2MASSJ0758+15301B_NIR_SpeX_Gagne2015c, chi_r^2 = 3.6, ndof=32
244/424: 2MASSJ1247-3816_NIR_SpeX_Gagne2014b, chi_r^2 = 9.2, ndof=39
245/424: SIMPJ0026-0936_NIR_SpeX_Robert2016, chi_r^2 = 8.3, ndof=39
246/424: 2MASSJ1005+1703_NIR_SpeX_Gagne2015c, chi_r^2 = 4.2, ndof=39
2MASSJ0017-3219_NIR_Flamingos-2_Gagne2015c.fits could not be opened. Corrupt file?
247/424 (2MASSJ0017-3219_NIR_Flamingos-2_Gagne2015c.fits) FAILED
248/424: SIMPJ1318-0632_NIR_SIMON_Robert2016, chi_r^2 = 15.1, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0945-7753_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
249/424 (2MASSJ0945-7753_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0240-4253_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
250/424 (2MASSJ0240-4253_NIR_Flamingos-2_Gagne2015c.fits) FAILED
251/424: SIMPJ2257-0140_NIR_SIMON_Robert2016, chi_r^2 = 96.7, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0126-5505_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
252/424 (2MASSJ0126-5505_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ1510-2818_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
253/424 (2MASSJ1510-2818_NIR_Flamingos-2_Gagne2015c.fits) FAILED
254/424: 2MASSJ1148-2836_NIR_SpeX_Gagne2015c, chi_r^2 = 15.1, ndof=39
255/424: 2MASSJ1028-2830_NIR_SpeX_Gagne2015c, chi_r^2 = 3.2, ndof=33
256/424: SIMPJ1344-1614_NIR_SpeX_Robert2016, chi_r^2 = 39.5, ndof=39
257/424: SIMPJ1122+0343_NIR_SIMON_Robert2016, chi_r^2 = 13.4, ndof=39
258/424: SIMPJ0103+1940_NIR_SpeX_Robert2016, chi_r^2 = 8.0, ndof=39
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template SIMPJ0417+1634_NIR_SpeX_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
259/424 (SIMPJ0417+1634_NIR_SpeX_Robert2016.fits) FAILED
260/424: SIMPJ1308+0432_NIR_SIMON_Robert2016, chi_r^2 = 15.7, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ1935-6200_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
261/424 (2MASSJ1935-6200_NIR_Flamingos-2_Gagne2015c.fits) FAILED
SIMPJ2344+0909_NIR_GNIRS_Robert2016.fits could not be opened. Corrupt file?
262/424 (SIMPJ2344+0909_NIR_GNIRS_Robert2016.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J19350976-6200473_ALLCOMB_merge' /                                ␣
↪[astropy.io.fits.card]
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2202-1109_merge'   /                                             ␣
↪[astropy.io.fits.card]
```

```
263/424: 2MASSJ2202-1109_NIR_SpeX_Gagne2015c, chi_r^2 = 5.5, ndof=32
2MASSJ0103-2805A_NIR_SpeX_Gagne2015c.fits could not be opened. Corrupt file?
264/424 (2MASSJ0103-2805A_NIR_SpeX_Gagne2015c.fits) FAILED
Unsufficient number of good points (33 < 34). Tmp discarded.
265/424 (2MASSJ0019+4614_NIR_SpeX_Gagne2015c.fits) FAILED
266/424: SIMPJ1537-0800_NIR_SIMON_Robert2016, chi_r^2 = 19.1, ndof=39
267/424: SIMPJ0858+2214_NIR_SIMON_Robert2016, chi_r^2 = 15.3, ndof=39
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template SIMPJ1654+3747_NIR_SpeX_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.43)mu?
268/424 (SIMPJ1654+3747_NIR_SpeX_Robert2016.fits) FAILED
269/424: SIMPJ1044+0620_NIR_SIMON_Robert2016, chi_r^2 = 12.4, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0210-3015_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
270/424 (2MASSJ0210-3015_NIR_Flamingos-2_Gagne2015c.fits) FAILED
271/424: SIMPJ2203-0301_NIR_SpeX_Robert2016, chi_r^2 = 10.6, ndof=39
272/424: SIMPJ0946+0922_NIR_GNIRS_Robert2016, chi_r^2 = 272.7, ndof=36
273/424: SIMPJ0307+0852_NIR_GNIRS_Robert2016, chi_r^2 = 207.5, ndof=38
Wavelength range of template 2MASSJ2336-3541_NIR_Flamingos-2_Gagne2015c.fits (0.90, 1.
↪89)mu too short compared to that of observed spectrum (0.95, 2.25)mu
274/424 (2MASSJ2336-3541_NIR_Flamingos-2_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2336-3541_BEAMS_AB_140710_397_398_399_400_comb.fits' /           ␣
↪[astropy.io.fits.card]
```

```
275/424: SIMPJ1036+0306_NIR_GNIRS_Robert2016, chi_r^2 = 63.0, ndof=38
No indices match the constraint (floor w.r.t 0.91)
Issue with resampling of template 2MASSJ2033-3733_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.91, 2.46)mu?
276/424 (2MASSJ2033-3733_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Unsufficient number of good points (33 < 34). Tmp discarded.
277/424 (2MASSJ1256-2718_NIR_FIRE_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2033-3733_ALLCOMB_merge' /                                       ␣
↪[astropy.io.fits.card]
```

```
278/424: SIMPJ1052+1722_NIR_GNIRS_Robert2016, chi_r^2 = 230.3, ndof=38
279/424: SIMPJ1605+1931_NIR_SpeX_Robert2016, chi_r^2 = 16.3, ndof=39
280/424: 2MASSJ0058-0651_NIR_SpeX_Gagne2015c, chi_r^2 = 11.5, ndof=32
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0708-4701_NIR_Flamingos-2_Gagne2015c.fits. Does␣
```

(continues on next page)

(continued from previous page)

```
↪the wavelength range extend far enough (0.89, 2.47)mu?
281/424 (2MASSJ0708-4701_NIR_Flamingos-2_Gagne2015c.fits) FAILED
282/424: SIMPJ0423+1212_NIR_GNIRS_Robert2016, chi_r^2 = 16.1, ndof=39
283/424: SIMPJ0422+0723_NIR_SpeX_Robert2016, chi_r^2 = 13.9, ndof=39
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ2033-5635_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.47)mu?
284/424 (2MASSJ2033-5635_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Unsufficient number of good points (32 < 34). Tmp discarded.
285/424 (2MASSJ2114-4339_NIR_SpeX_Gagne2015c.fits) FAILED

WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2114-4339_merge'   /                                              ␣
↪[astropy.io.fits.card]

286/424: SIMPJ0915+0514_NIR_SpeX_Robert2016, chi_r^2 = 14.1, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2050-3639_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
287/424 (2MASSJ2050-3639_NIR_Flamingos-2_Gagne2015c.fits) FAILED

WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J20505221-3639552_ALLCOMB_merge' /                                 ␣
↪[astropy.io.fits.card]

288/424: SIMPJ2304+0749_NIR_GNIRS_Robert2016, chi_r^2 = 16.5, ndof=36
289/424: SIMPJ0136+0933_NIR_GNIRS_Robert2016, chi_r^2 = 79.7, ndof=36
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ0034-4102_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.47)mu?
290/424 (2MASSJ0034-4102_NIR_Flamingos-2_Gagne2015c.fits) FAILED
291/424: SIMPJ1627+0546_NIR_SIMON_Robert2016, chi_r^2 = 12.1, ndof=39
292/424: SIMPJ0013-1816_NIR_SpeX_Robert2016, chi_r^2 = 23.4, ndof=39
293/424: SIMPJ2330+1006_NIR_GNIRS_Robert2016, chi_r^2 = 297.0, ndof=38
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2322-6151A_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
294/424 (2MASSJ2322-6151A_NIR_Flamingos-2_Gagne2015c.fits) FAILED

WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2322-6151_ALLCOMB_merge' /                                        ␣
↪[astropy.io.fits.card]

295/424: SIMPJ1138+0748_NIR_SIMON_Robert2016, chi_r^2 = 48.5, ndof=39
296/424: 2MASSJ0548-2942_NIR_SpeX_Gagne2015c, chi_r^2 = 6.6, ndof=36
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ1622-2346_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
297/424 (2MASSJ1622-2346_NIR_SpeX_Gagne2015c.fits) FAILED
298/424: SIMPJ1343+1312_NIR_GNIRS_Robert2016, chi_r^2 = 14.8, ndof=39
Wavelength range of template SIMPJ0006-2158_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
```

(continues on next page)

(continued from previous page)

```
299/424 (SIMPJ0006-2158_NIR_NIRI_Robert2016.fits) FAILED
Unsufficient number of good points (33 < 34). Tmp discarded.
300/424 (2MASSJ2353-1844B_NIR_SpeX_Gagne2015c.fits) FAILED
Wavelength range of template SIMPJ1615+1340_NIR_GNIRS_Robert2016.fits (1.00, 2.52)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
301/424 (SIMPJ1615+1340_NIR_GNIRS_Robert2016.fits) FAILED
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ0501-0010_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
302/424 (2MASSJ0501-0010_NIR_SpeX_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2329_tell.fits'    /                                                    ␣
↪[astropy.io.fits.card]
```

```
303/424: 2MASSJ2329+0329_NIR_SpeX_Gagne2015c, chi_r^2 = 7.9, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0540-0906_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
304/424 (2MASSJ0540-0906_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Wavelength range of template DENISJ0817-6155_NIR_OSIRIS_Artigau2010.fits (1.13, 2.36)mu␣
↪too short compared to that of observed spectrum (0.95, 2.25)mu
305/424 (DENISJ0817-6155_NIR_OSIRIS_Artigau2010.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0228+0218_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
306/424 (2MASSJ0228+0218_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ1051-1916_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
307/424 (2MASSJ1051-1916_NIR_SpeX_Gagne2015c.fits) FAILED
SIMPJ2351+3010_NIR_SpeX_Robert2016.fits could not be opened. Corrupt file?
308/424 (SIMPJ2351+3010_NIR_SpeX_Robert2016.fits) FAILED
309/424: SIMPJ1339+1523_NIR_GNIRS_Robert2016, chi_r^2 = 283.0, ndof=37
310/424: SIMPJ0812+3721_NIR_SpeX_Robert2016, chi_r^2 = 9.6, ndof=39
311/424: SIMPJ1422+0827_NIR_GNIRS_Robert2016, chi_r^2 = 122.3, ndof=38
312/424: SIMPJ2143-1544_NIR_SpeX_Robert2016, chi_r^2 = 9.1, ndof=39
313/424: SIMPJ2245+1722_NIR_SIMON_Robert2016, chi_r^2 = 19.9, ndof=39
314/424: GUPscA_NIR_SpeX_Naud2014, chi_r^2 = 3.3, ndof=39
Unsufficient number of good points (33 < 34). Tmp discarded.
315/424 (2MASSJ1839+2952_NIR_SpeX_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ2002-1316_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
316/424 (2MASSJ2002-1316_NIR_SpeX_Gagne2015c.fits) FAILED
```

```
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J1839+2952_merge'   /                                                    ␣
↪[astropy.io.fits.card]
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
```

(continues on next page)

(continued from previous page)

```
I_R_A_FNAME= 'J2002-1316_merge'   /                                             ␣
↪[astropy.io.fits.card]
317/424: SIMPJ1147-1032_NIR_GNIRS_Robert2016, chi_r^2 = 226.5, ndof=39
Wavelength range of template SIMPJ1039+2440_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
318/424 (SIMPJ1039+2440_NIR_NIRI_Robert2016.fits) FAILED
319/424: 2MASSJ0344+0716A_NIR_SpeX_Gagne2015c, chi_r^2 = 3.1, ndof=33
Wavelength range of template SIMPJ0004-2007_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
320/424 (SIMPJ0004-2007_NIR_NIRI_Robert2016.fits) FAILED
321/424: 2MASSJ0336-2619_NIR_SpeX_Gagne2015c, chi_r^2 = 4.8, ndof=39
322/424: SIMPJ0946+1808_NIR_SIMON_Robert2016, chi_r^2 = 11.3, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0545-0121_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
323/424 (2MASSJ0545-0121_NIR_Flamingos-2_Gagne2015c.fits) FAILED
324/424: SIMPJ1003-1441_NIR_SIMON_Robert2016, chi_r^2 = 7.4, ndof=39
325/424: SIMPJ1308+0432_NIR_GNIRS_Robert2016, chi_r^2 = 290.2, ndof=37
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template SIMPJ1623-0508_NIR_SpeX_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.43)mu?
326/424 (SIMPJ1623-0508_NIR_SpeX_Robert2016.fits) FAILED
327/424: CFBDSIR2149_OPT+NIR_X-Shooter_Delorme2012, chi_r^2 = 166.3, ndof=39
Wavelength range of template SIMPJ1533+2044_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
328/424 (SIMPJ1533+2044_NIR_NIRI_Robert2016.fits) FAILED
329/424: SIMPJ1200-2836_NIR_SpeX_Robert2016, chi_r^2 = 30.8, ndof=39
330/424: SIMPJ2250+0808_NIR_SpeX_Robert2016, chi_r^2 = 28.4, ndof=39
331/424: SIMPJ1510-1147_NIR_SIMON_Robert2016, chi_r^2 = 19.9, ndof=39
332/424: 2MASSJ0814+0253_NIR_SpeX_Gagne2015c, chi_r^2 = 3.4, ndof=32
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ1200-3405_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
333/424 (2MASSJ1200-3405_NIR_Flamingos-2_Gagne2015c.fits) FAILED
334/424: SIMPJ1035+3655_NIR_SpeX_Robert2016, chi_r^2 = 7.8, ndof=39
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2039-1126_merge'   /                                             ␣
↪[astropy.io.fits.card]
335/424: 2MASSJ2039-1126_NIR_SpeX_Gagne2015c, chi_r^2 = 6.6, ndof=35
336/424: 2MASSJ0803+0827_NIR_SpeX_Gagne2015c, chi_r^2 = 3.8, ndof=33
337/424: SIMPJ2157-0130_NIR_SIMON_Robert2016, chi_r^2 = 19.1, ndof=39
338/424: 2MASSJ0805+2505A_NIR_SpeX_Gagne2015c, chi_r^2 = 3.8, ndof=39
339/424: SIMPJ1256-1002_NIR_SpeX_Robert2016, chi_r^2 = 7.7, ndof=39
340/424: SIMPJ1118-0856_NIR_SpeX_Robert2016, chi_r^2 = 26.9, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2313-6127_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
341/424 (2MASSJ2313-6127_NIR_Flamingos-2_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
```

(continues on next page)

(continued from previous page)

```
↪convention:
I_R_A_FNAME= 'J2313-6127_ALLCOMB_merge' /                                       ␣
↪[astropy.io.fits.card]
342/424: SIMPJ0410+1459_NIR_SpeX_Robert2016, chi_r^2 = 12.0, ndof=39
343/424: SIMPJ1009+1559_NIR_GNIRS_Robert2016, chi_r^2 = 254.7, ndof=35
344/424: SIMPJ2213+1255_NIR_SpeX_Robert2016, chi_r^2 = 7.0, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0027-0806_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
345/424 (2MASSJ0027-0806_NIR_Flamingos-2_Gagne2015c.fits) FAILED
346/424: 2MASSJ0507+1430B_NIR_SpeX_Gagne2015c, chi_r^2 = 3.3, ndof=39
347/424: SIMPJ0138-0322_NIR_SIMON_Robert2016, chi_r^2 = 64.2, ndof=39
348/424: SIMPJ0155+0950_NIR_SIMON_Robert2016, chi_r^2 = 18.1, ndof=39
No indices match the constraint (floor w.r.t 0.85)
Issue with resampling of template 2MASSJ1252-3415_NIR_GNIRS_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.85, 2.52)mu?
349/424 (2MASSJ1252-3415_NIR_GNIRS_Gagne2015c.fits) FAILED
350/424: 2MASSJ0153-6744_NIR_FIRE_Gagne2015c, chi_r^2 = 12.5, ndof=32
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2322-6151B_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
351/424 (2MASSJ2322-6151B_NIR_Flamingos-2_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2322-6151B_ALLCOMB_merge' /                                      ␣
↪[astropy.io.fits.card]
352/424: SIMPJ1811+2728_NIR_SpeX_Robert2016, chi_r^2 = 21.2, ndof=39
No indices match the constraint (floor w.r.t 0.84)
Issue with resampling of template 2MASSJ1627-2411_NIR_GNIRS_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.84, 2.53)mu?
353/424 (2MASSJ1627-2411_NIR_GNIRS_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_F-TLM= '2013-07-18T20:09:23' / Time of last modification                  ␣
↪[astropy.io.fits.card]
354/424: 2MASSJ0021-4244_NIR_SpeX_Gagne2015c, chi_r^2 = 13.1, ndof=34
Unsufficient number of good points (33 < 34). Tmp discarded.
355/424 (2MASSJ2206-6116_NIR_FIRE_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'W2206-6116_tc.fits' /                                             ␣
↪[astropy.io.fits.card]
356/424: SIMPJ2255-0725_NIR_SIMON_Robert2016, chi_r^2 = 38.5, ndof=39
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template SIMPJ1455+2619_NIR_SpeX_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.43)mu?
357/424 (SIMPJ1455+2619_NIR_SpeX_Robert2016.fits) FAILED
Unsufficient number of good points (29 < 34). Tmp discarded.
358/424 (2MASSJ0258-1520_NIR_FIRE_Gagne2015c.fits) FAILED
```

(continues on next page)

(continued from previous page)

```
359/424: SIMPJ1127+1234_NIR_GNIRS_Robert2016, chi_r^2 = 117.2, ndof=36
Wavelength range of template SIMPJ1756+3343_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
360/424 (SIMPJ1756+3343_NIR_NIRI_Robert2016.fits) FAILED
No indices match the constraint (floor w.r.t 0.85)
Issue with resampling of template SIMPJ0358+1039_NIR_SIMON_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.85, 2.44)mu?
361/424 (SIMPJ0358+1039_NIR_SIMON_Robert2016.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2257-5041_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
362/424 (2MASSJ2257-5041_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template 2MASSJ1301-1510_NIR_SpeX_Gagne2015c.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
363/424 (2MASSJ1301-1510_NIR_SpeX_Gagne2015c.fits) FAILED

WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2257-5041_ALLCOMB_merge' /                                       ␣
↪[astropy.io.fits.card]

364/424: SIMPJ1130+2341_NIR_SpeX_Robert2016, chi_r^2 = 25.9, ndof=39
365/424: SIMPJ0357-1937_NIR_SpeX_Robert2016, chi_r^2 = 8.9, ndof=39
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ2154-7459_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.47)mu?
366/424 (2MASSJ2154-7459_NIR_Flamingos-2_Gagne2015c.fits) FAILED

WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2154-7459_ALLCOMB_merge' /                                       ␣
↪[astropy.io.fits.card]

367/424: 2MASSJ2206-4217_NIR_SpeX_Gagne2015c, chi_r^2 = 17.3, ndof=39
Wavelength range of template SIMPJ1143+1905_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
368/424 (SIMPJ1143+1905_NIR_NIRI_Robert2016.fits) FAILED
Wavelength range of template SIMPJ2315+2235_NIR_NIRI_Robert2016.fits (0.98, 2.45)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
369/424 (SIMPJ2315+2235_NIR_NIRI_Robert2016.fits) FAILED
370/424: SIMPJ2248-0126_NIR_SpeX_Robert2016, chi_r^2 = 17.6, ndof=39
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ1542-3358_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.46)mu?
371/424 (2MASSJ1542-3358_NIR_Flamingos-2_Gagne2015c.fits) FAILED
372/424: SIMPJ0432-0639_NIR_SpeX_Robert2016, chi_r^2 = 11.7, ndof=39
373/424: SIMPJ1644+2600_NIR_SpeX_Robert2016, chi_r^2 = 16.3, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0006-6436_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
374/424 (2MASSJ0006-6436_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0253-7959_NIR_Flamingos-2_Gagne2015c.fits. Does␣
```

(continues on next page)

(continued from previous page)

```
↪the wavelength range extend far enough (0.89, 2.47)mu?
375/424 (2MASSJ0253-7959_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ0241-5511_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.47)mu?
376/424 (2MASSJ0241-5511_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Wavelength range of template SIMPJ0102+0355_NIR_GNIRS_Robert2016.fits (1.00, 2.52)mu too␣
↪short compared to that of observed spectrum (0.95, 2.25)mu
377/424 (SIMPJ0102+0355_NIR_GNIRS_Robert2016.fits) FAILED
378/424: 2MASSJ0449+1607_NIR_SpeX_Gagne2015c, chi_r^2 = 15.5, ndof=39
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template SIMPJ0940+2946_NIR_SpeX_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
379/424 (SIMPJ0940+2946_NIR_SpeX_Robert2016.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0512-3041_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
380/424 (2MASSJ0512-3041_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0250-0151_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
381/424 (2MASSJ0250-0151_NIR_Flamingos-2_Gagne2015c.fits) FAILED
382/424: SIMPJ0155+0950_NIR_GNIRS_Robert2016, chi_r^2 = 26.4, ndof=39
Unsufficient number of good points (33 < 34). Tmp discarded.
383/424 (2MASSJ0046+0252_NIR_SpeX_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template SIMPJ0956-1910_NIR_SpeX_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.43)mu?
384/424 (SIMPJ0956-1910_NIR_SpeX_Robert2016.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2038-4118_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
385/424 (2MASSJ2038-4118_NIR_Flamingos-2_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2038-4118_ALLCOMB_merge' /                                              ␣
↪[astropy.io.fits.card]
386/424: SIMPJ1013-1946_NIR_GNIRS_Robert2016, chi_r^2 = 270.5, ndof=37
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2314-5405_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
387/424 (2MASSJ2314-5405_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0440-1820_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
388/424 (2MASSJ0440-1820_NIR_Flamingos-2_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2314-5405_ALLCOMB_merge' /                                              ␣
↪[astropy.io.fits.card]
```

```
389/424: SIMPJ1108+0838_NIR_GNIRS_Robert2016, chi_r^2 = 43.2, ndof=36
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0004-1709_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
390/424 (2MASSJ0004-1709_NIR_Flamingos-2_Gagne2015c.fits) FAILED
391/424: 2MASSJ1325+0600_NIR_SpeX_Gagne2015c, chi_r^2 = 11.4, ndof=39
392/424: SIMPJ2235+0418_NIR_SIMON_Robert2016, chi_r^2 = 18.9, ndof=39
2MASSJ0017-0316_NIR_SpeX_Gagne2015c.fits could not be opened. Corrupt file?
393/424 (2MASSJ0017-0316_NIR_SpeX_Gagne2015c.fits) FAILED
394/424: SIMPJ0421-1950_NIR_GNIRS_Robert2016, chi_r^2 = 195.8, ndof=35
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0417-1140_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
395/424 (2MASSJ0417-1140_NIR_Flamingos-2_Gagne2015c.fits) FAILED
396/424: 2MASSJ0541-0737_NIR_SpeX_Gagne2015c, chi_r^2 = 12.9, ndof=32
No indices match the constraint (floor w.r.t 0.92)
Issue with resampling of template 2MASSJ2134-1840_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.92, 2.46)mu?
397/424 (2MASSJ2134-1840_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Wavelength range of template 2MASSJ1253-4211_NIR_Flamingos-2_Gagne2015c.fits (0.90, 1.
↪89)mu too short compared to that of observed spectrum (0.95, 2.25)mu
398/424 (2MASSJ1253-4211_NIR_Flamingos-2_Gagne2015c.fits) FAILED
Unsufficient number of good points (29 < 34). Tmp discarded.
399/424 (2MASSJ1633-2425_NIR_SpeX_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2134-1840_ALLCOMB_merge' /                                       ␣
↪[astropy.io.fits.card]
400/424: SIMPJ1102+1629_NIR_GNIRS_Robert2016, chi_r^2 = 73.6, ndof=36
401/424: SIMPJ0945-0757_NIR_SpeX_Robert2016, chi_r^2 = 17.4, ndof=39
402/424: SIMPJ2322+1300_NIR_SIMON_Robert2016, chi_r^2 = 18.7, ndof=39
No indices match the constraint (floor w.r.t 0.90)
Issue with resampling of template 2MASSJ0318-3708_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.90, 2.47)mu?
403/424 (2MASSJ0318-3708_NIR_Flamingos-2_Gagne2015c.fits) FAILED
404/424: SIMPJ1343-1216_NIR_SpeX_Robert2016, chi_r^2 = 35.3, ndof=39
405/424: 2MASSJ0527+0007A_NIR_SpeX_Gagne2015c, chi_r^2 = 7.3, ndof=39
406/424: SIMPJ1218-1332_NIR_GNIRS_Robert2016, chi_r^2 = 293.4, ndof=36
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0854-3051_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
407/424 (2MASSJ0854-3051_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0720-5617_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
408/424 (2MASSJ0720-5617_NIR_Flamingos-2_Gagne2015c.fits) FAILED
409/424: SIMPJ0006-0852_NIR_GNIRS_Robert2016, chi_r^2 = 19.8, ndof=39
410/424: SIMPJ0331+1944_NIR_SIMON_Robert2016, chi_r^2 = 29.3, ndof=39
2MASSJ0002+0408B_NIR_SpeX_Gagne2015c.fits could not be opened. Corrupt file?
411/424 (2MASSJ0002+0408B_NIR_SpeX_Gagne2015c.fits) FAILED
412/424: 2MASSJ0524+0640_NIR_SpeX_Gagne2015c, chi_r^2 = 4.2, ndof=35
No indices match the constraint (floor w.r.t 0.89)
```

(continues on next page)

(continued from previous page)

```
Issue with resampling of template 2MASSJ0440-5126_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
413/424 (2MASSJ0440-5126_NIR_Flamingos-2_Gagne2015c.fits) FAILED
414/424: 2MASSJ0720-0846_NIR_SpeX_Gagne2015c, chi_r^2 = 13.3, ndof=39
415/424: SIMPJ1118-0856_NIR_SIMON_Robert2016, chi_r^2 = 17.8, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0134-5707_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
416/424 (2MASSJ0134-5707_NIR_Flamingos-2_Gagne2015c.fits) FAILED
417/424: SIMPJ2233-1331_NIR_SIMON_Robert2016, chi_r^2 = 43.7, ndof=39
418/424: 2MASSJ0333-3215_NIR_SpeX_Gagne2015c, chi_r^2 = 4.1, ndof=33
419/424: SIMPJ0430+1035_NIR_GNIRS_Robert2016, chi_r^2 = 254.0, ndof=39
No indices match the constraint (floor w.r.t 0.94)
Issue with resampling of template SIMPJ0151+3824_NIR_SpeX_Robert2016.fits. Does the␣
↪wavelength range extend far enough (0.94, 2.42)mu?
420/424 (SIMPJ0151+3824_NIR_SpeX_Robert2016.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ2022-5645_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.46)mu?
421/424 (2MASSJ2022-5645_NIR_Flamingos-2_Gagne2015c.fits) FAILED
WARNING: The following header keyword is invalid or follows an unrecognized non-standard␣
↪convention:
I_R_A_FNAME= 'J2022-5645_ALLCOMB_merge' /                                       ␣
↪[astropy.io.fits.card]
422/424: 2MASSJ1247-3816_NIR_SpeX_Gagne2015c, chi_r^2 = 9.2, ndof=39
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0418-4507_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
423/424 (2MASSJ0418-4507_NIR_Flamingos-2_Gagne2015c.fits) FAILED
No indices match the constraint (floor w.r.t 0.89)
Issue with resampling of template 2MASSJ0423-1533_NIR_Flamingos-2_Gagne2015c.fits. Does␣
↪the wavelength range extend far enough (0.89, 2.47)mu?
424/424 (2MASSJ0423-1533_NIR_Flamingos-2_Gagne2015c.fits) FAILED
**************************************

228/424 template spectra were fitted.

Running time:  0:08:56.400537
-------------------------------------------------------------------------------
best chi:  [2.6494487  2.71261062 2.87087266]
best scale fac:  [8.93336943e-05 5.61832126e-05 9.14677818e-03]
n_dof:  [33. 33. 32.]
The best template #1 is: 2MASSJ1119-3917_NIR_SpeX_Gagne2015c.fits (Delta A_V=2.4mag)

The best template #2 is: 2MASSJ1153-3015_NIR_SpeX_Gagne2015c.fits (Delta A_V=2.4mag)

The best template #3 is: 2MASSJ0758+1530O_NIR_SpeX_Gagne2015c.fits (Delta A_V=2.2mag)
```

Let's load the results, and prepare a list of short template names (for display; i.e. without the suffix):

[141]:
```
final_tmpname, final_tmp, final_chi, final_params, final_ndof = final_res

n_end = len('_NIR_SpeX_Gagne2015c.fits')
short_tmpname = [final_tmpname[i][:-n_end] for i in range(len(final_tmpname))]
short_tmpname
```

[141]:
```
['2MASSJ1119-3917', '2MASSJ1153-3015', '2MASSJ0758+15300']
```

Checking in the online table of the MSL, these spectra turn out to correspond to the following types of objects:

[142]:
```
labels_tmp = ["M3", "M4.5", "M4"]
```

Feel free to repeat the fit after changing the exact value of `min_npts` (e.g.~lower this value to allow more templates to be considered in the search). Although the best-fit templates may change a little, you should mostly retrieve M3-M5 spectral types as top-3 template spectra for reasonable values of `min_npts`.

Let's finally plot the best-fit template spectra, at native resolution and resampled, respectively:

[143]:
```
from special.model_resampling import resample_model

# Prepare plot
fig, (ax1, ax2) = plt.subplots(2,1,figsize=(11,12))

ax1.set_xlabel(r"Wavelength ($\mu$m)")
ax1.set_ylabel("$\lambda F_{\lambda}$ (W m$^{-2}$)")
ax1.set_ylim(0.8*np.amin(lbda*(spec+spec_err)),
             1.1*np.amax(lbda*(spec+spec_err)))

ax2.set_xlabel(r"Wavelength ($\mu$m)")
ax2.set_ylabel("$\lambda F_{\lambda}$ (W m$^{-2}$)")
ax2.set_ylim(0.8*np.amin(lbda*(spec+spec_err)),
             1.1*np.amax(lbda*(spec+spec_err)))

# plot options
cols = ['k', 'r', 'b', 'c', 'y', 'm'] # colors of different models
maj_tick_sp = 0.5 # WL spacing for major ticks
min_tick_sp = maj_tick_sp / 5. # WL spacing for minor ticks

# Plot of top 3 spectra:
counter = 0
object_list = []
for ll in range(n_best):
    ## loading
    try:
        _, header = open_fits(inpath_models+final_tmpname[ll],header=True)
        object_list.append(header['OBJECT'])
    except KeyError:
        object_list.append(short_tmpname[ll])
    lbda_tmp, flux_tmp, flux_tmp_err = final_tmp[ll]

    ## cropping range
    try:
        idx_ini = find_nearest(lbda_tmp,0.98*lbda_crop[0], constraint='floor')
    except:
```

(continues on next page)

(continued from previous page)

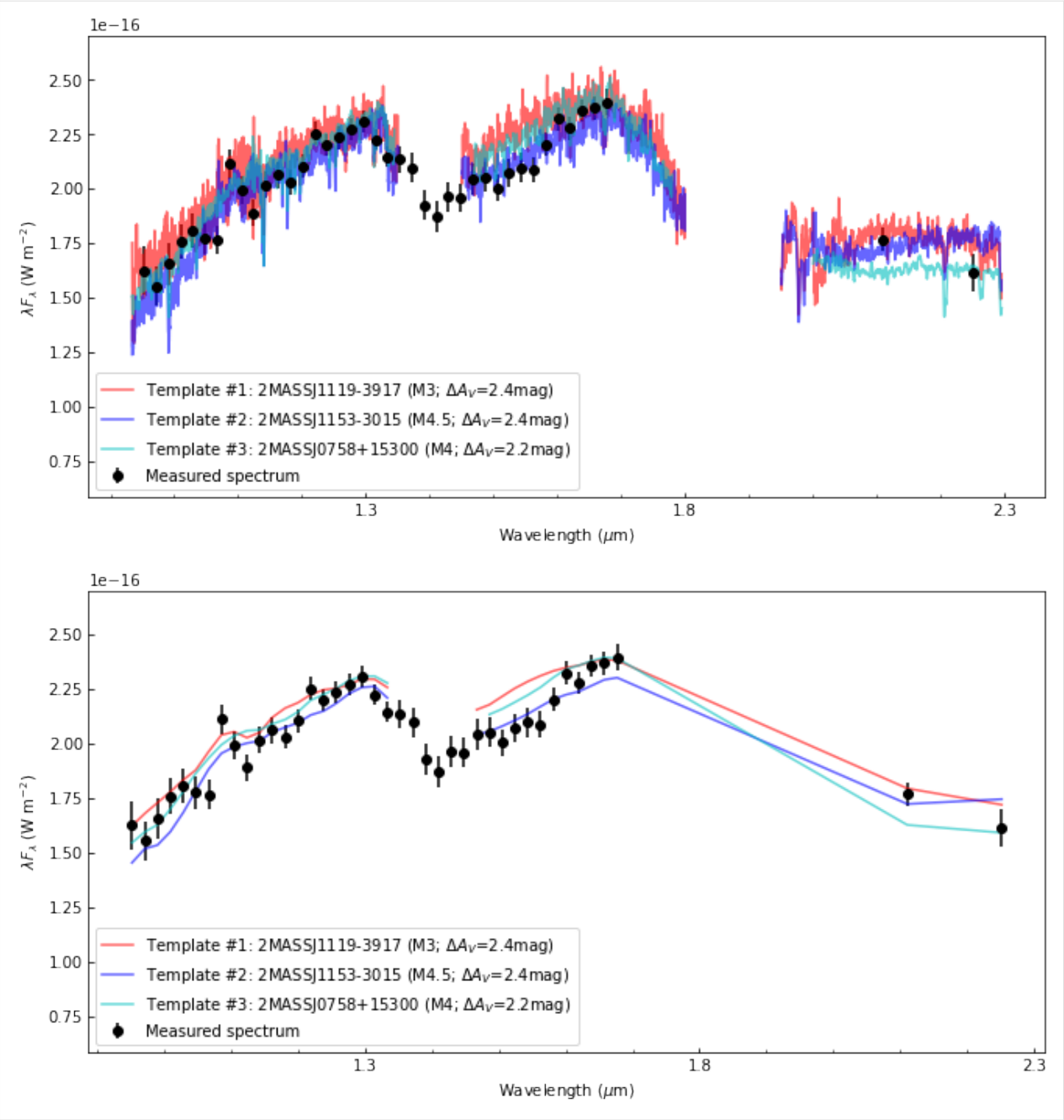
```
        idx_ini = 0
    try:
        idx_fin = find_nearest(lbda_tmp,1.02*lbda_crop[-1], constraint='ceil')
    except:
        idx_fin = -1
    lbda_tmp_crop = lbda_tmp[idx_ini:idx_fin]
    flux_tmp_crop = flux_tmp[idx_ini:idx_fin]
    dlbda_tmp = np.mean(lbda_tmp_crop[1:]-lbda_tmp_crop[:-1])

    ## Convolving + resampling template spectrum
    tmp_res = resample_model(lbda_crop, lbda_tmp, flux_tmp,
                             dlbda_obs=dlbda_crop,
                             instru_res=instru_res_crop,
                             instru_idx=instru_idx_crop,
                             filter_reader=filter_reader)
    lbda_tmp_res, flux_tmp_res = tmp_res

    ## formatting labels
    lab_str_list = [labels_tmp[ll]]
    if AV_range is not None:
        lab_str = r'$\Delta A_V$={0:.{1}f}{2}'.format(final_params[1][ll],1,'mag')
        lab_str_list.append(lab_str)
    sep = '; '
    if n_best>1:
        label = "Template #{:.0f}: {} ({})".format(ll+1, short_tmpname[ll], #labels_
↪tmp[ll],
                                                   sep.join(lab_str_list))
    else:
        label = "Best template: {} ({})".format(short_tmpname[ll], #labels_tmp[ll],
                                                sep.join(lab_str_list))

    ## plotting
    ax1.plot(lbda_tmp_crop, lbda_tmp_crop*flux_tmp_crop, cols[ll+1],
             alpha=0.6, label=label)
    ax2.plot(lbda_tmp_res, lbda_tmp_res*flux_tmp_res, cols[ll+1],
             alpha=0.6, label=label)

    ## ticks
    min_tick = lbda_crop[0]-dlbda_crop[0]/2-((lbda_crop[0]-dlbda_crop[0]/2)%0.2)
    max_tick = lbda_crop[-1]+dlbda_crop[-1]/2+(0.2-((lbda_crop[-1]+dlbda_crop[-1]/2)%0.
↪2))
    major_ticks1 = np.arange(min_tick,max_tick,maj_tick_sp)
    minor_ticks1 = np.arange(min_tick,max_tick,min_tick_sp)

    ax1.set_xticks(major_ticks1)
    ax1.set_xticks(minor_ticks1, minor = True)
    ax1.tick_params(which = 'both', direction = 'in')
    ax2.set_xticks(major_ticks1)
    ax2.set_xticks(minor_ticks1, minor = True)
    ax2.tick_params(which = 'both', direction = 'in')

# Plot measured spectrum
```

(continues on next page)

(continued from previous page)

```
ax1.errorbar(lbda_crop, lbda_crop*spec_crop,
             lbda_crop*spec_err_crop, fmt=cols[0]+'o',
             label='Measured spectrum')
ax1.legend(loc='best')
ax2.errorbar(lbda_crop, lbda_crop*spec_crop,
             lbda_crop*spec_err_crop, fmt=cols[0]+'o',
             label='Measured spectrum')
ax2.legend(loc='best')

#plt.savefig("Template_fit.pdf", bbox_inches='tight')
plt.show()
```

```
Fits HDU-0 data and header successfully loaded. Data shape: (3, 4142)
Fits HDU-0 data and header successfully loaded. Data shape: (3, 4145)
Fits HDU-0 data and header successfully loaded. Data shape: (3, 4153)
```


1e−16
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
λF_λ (W m^{-2})
Template #1: 2MASSJ1119-3917 (M3; ΔA_V=2.4mag)
Template #2: 2MASSJ1153-3015 (M4.5; ΔA_V=2.4mag)
Template #3: 2MASSJ0758+15300 (M4; ΔA_V=2.2mag)
Measured spectrum
1.3
1.8
2.3
Wavelength (μm)
1e−16
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
λF_λ (W m^{-2})
Template #1: 2MASSJ1119-3917 (M3; ΔA_V=2.4mag)
Template #2: 2MASSJ1153-3015 (M4.5; ΔA_V=2.4mag)
Template #3: 2MASSJ0758+15300 (M4; ΔA_V=2.2mag)
Measured spectrum
1.3
1.8
2.3
Wavelength (μm)

Go to the top

CHAPTER

TWELVE

CONTRIBUTIONS

Feel free to fork the repository and submit a pull request with either new features or bug fixes. External contributions are very welcome. In particular, please check the expected future areas for development.

CHAPTER

THIRTEEN

QUESTIONS AND SUGGESTIONS

`special` was developed by Valentin Christiaens. Feel free to contact me at valentin.christiaens@uliege.be if you have any question or suggestion.

CHAPTER

FOURTEEN

ACKNOWLEDGEMENTS

Please cite Christiaens et al. (2022) if you use `special` for your research, along with (where relevant):

- Foreman-Mackey et al. (2013) if you use the `emcee` MCMC sampler;
- Skilling (2004), Mukherjee et al. (2006), or Feroz et al. (2009) if you use the nested sampler *nestle* in 'classic', 'single' or 'multi' mode, respectively. Please also mention the `nestle` GitHub repository;
- Buchner (2021) if you use the *UltraNest* nested sampler.

CHAPTER

FIFTEEN

SPECIAL PACKAGE

15.1 Submodules

15.2 special.chi module

Function defining the goodness of fit.

special.chi.**gof_scal**(*params*, *lbda_obs*, *spec_obs*, *err_obs*, *lbda_tmp*, *spec_tmp*, *dlbda_obs*, *instru_corr*, *instru_res*, *instru_idx*, *use_weights*, *filter_reader*, *ext_range*)

Wrapper of routine `special.chi.goodness_of_fit` for the goodness of fit used to search for the best-fit library template spectrum to an observed spectrum. It has a `params` argument to consider the scaling factor and optionnally the differential extinction as free parameter(s).

Parameters

- **params** (*tuple*) – Tuple of 1 or 2 elements: scaling factor and (optionally) differential optical extinction ΔA_V (ΔA_V can be negative if template spectra are not dereddened).
- **lbda_obs** (*numpy 1d ndarray or list*) – Wavelengths of observed spectrum. If several instruments were used, the wavelengths should be ordered per instrument, not necessarily as monotonically increasing wavelength. Hereafter, n_{ch} is the length of `lbda_obs`.
- **spec_obs** (*numpy 1d ndarray or list*) – Observed spectrum for each value of `lbda_obs`. Should have a length of n_{ch}.
- **err_obs** (*numpy 1d/2d ndarray or list*) – Uncertainties on the observed spectrum. The array (list) can have either a length of n_{ch}, or a shape of $(2, n_{ch})$ for lower (first column) and upper (second column) uncertainties provided.
- **lbda_tmp** (*numpy 1d ndarray or list*) – Wavelengths of tested template. Should have a wider wavelength extent than the observed spectrum.
- **spec_tmp** (*numpy 1d ndarray*) – Template spectrum. It does not require the same wavelength sampling as the observed spectrum. If higher spectral resolution, it will be convolved with the instrumental spectral PSF (if `instru_res` is provided) and then binned to the same sampling. If lower spectral resolution, a linear interpolation is performed to infer the value at the observed spectrum wavelength sampling.
- **dlbda_obs** (*numpy 1d ndarray or list, optional*) – Respective spectral channel width or FWHM of the photometric filters used for each point of the observed spectrum. This vector is used to infer which part(s) of a combined spectro+photometric spectrum should involve convolution+subsampling (model resolution higher than measurements), interpolation (the opposite), or convolution by the transmission curve of a photometric filter. If not provided, it will be inferred from the difference between consecutive lbda_obs points (i.e.

inaccurate for a combined spectrum). It must be provided IF one wants to weigh each measurement based on the spectral resolution of each instrument (as in [OLO16]), through the `use_weights` argument.

- **instru_corr** (*numpy 2d ndarray, optional*) – Spectral correlation between post-processed images in which the spectrum is measured. It is specific to the instrument, PSF subtraction algorithm and radial separation of the companion from the central star. Can be computed using `special.spec_corr.spectral_correlation`. In case of a spectrum obtained with different instruments, it is recommended to construct the final spectral_correlation matrix with `special.spec_corr.combine_corrs`. If `instru_corr` is not provided, the uncertainties in each spectral channel will be considered independent. See [GRE16] for more details.
- **instru_res** (*float or list of floats/strings, optional*) – The mean instrumental resolving power(s) OR filter names. This is used to convolve the model spectrum. If several instruments are used, provide a list of resolving power values / filter names, one for each instrument used.
- **instru_idx** (*numpy 1d array, optional*) – 1d array containing an index representing each instrument used to obtain the spectrum, label them from 0 to the number of instruments (n_{ins}). Zero for points that don't correspond to any of the `instru_res` values provided, and i in $[1, n_{ins}]$ for points associated to instru_res[i-1]. This parameter must be provided if the spectrum consists of points obtained with different instruments.
- **use_weights** (*bool, optional*) – For the likelihood calculation, whether to weigh each point of the spectrum based on the spectral resolution or bandwidth of photometric filters used. Weights will be proportional to `dlbda_obs/lbda_obs` if `dlbda_obs` is provided, or set to 1 for all points otherwise.
- **filter_reader** (*python routine, optional*) – External routine that reads a filter file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains transmission values. Important: if not provided, but strings are detected in instru_res, the default file reader will be used. It assumes the following format for the files:
  - first row contains headers (titles of each column)
  - starting from 2nd row: 1st column: wavelength, 2nd col.: transmission
  - Unit of wavelength can be provided in parentheses of first header key name: e.g. "WL(AA)" for angstrom, "wavelength(mu)" for micrometer or "lambda(nm)" for nanometer. Note: only what is in parentheses matters for the units.
- **ext_range** (*tuple or None, opt*) –
  - If None: differential extinction is not considered as a free parameter.
  - If a tuple: it should contain 2 floats (for simplex `search_mode`) or 3 floats (for grid search `search_mode`) corresponding to the lower limit, upper limit (and step for the grid search). For the simplex search, the lower and upper limits are used to set a chi square of infinity outside of the range.

Returns
chi_sq – Goodness of fit indicator.

Return type
float

Note: If several filter filenames are provided in `instru_res`, the filter files must all have the same format and wavelength units (for reading by the same `filter_reader` snippet or default function).

---

special.chi.**goodness_of_fit**(*lbda_obs*, *spec_obs*, *err_obs*, *lbda_mod*, *spec_mod*, *dlbda_obs=None*, *instru_corr=None*, *instru_res=None*, *instru_idx=None*, *use_weights=True*, *filter_reader=None*, *plot=False*, *outfile=None*)

Function to estimate the goodness of fit indicator defined as in [OLO16] (Eq. 8). In addition, if a spectral correlation matrix is provided, it is used to take into account the correlated noise between spectral channels (see [GRE16]). The goodness of fit indicator is identical to a χ^2 if all points are obtained with the same instrument (or `use_weights` set to False) and in absence of spectrally correlated noise.

Parameters

- **lbda_obs** (*numpy 1d ndarray or list*) – Wavelengths of observed spectrum. If several instruments were used, the wavelengths should be ordered per instrument, not necessarily as monotonically increasing wavelength. Hereafter, n_{ch} is the length of `lbda_obs`.
- **spec_obs** (*numpy 1d ndarray or list*) – Observed spectrum for each value of `lbda_obs`. Should have a length of n_{ch}.
- **err_obs** (*numpy 1d/2d ndarray or list*) – Uncertainties on the observed spectrum. The array (list) can have either a length of n_{ch}, or a shape of $(2, n_{ch})$ for lower (first column) and upper (second column) uncertainties provided.
- **lbda_mod** (*numpy 1d ndarray or list*) – Wavelengths of tested model. Should have a wider (or equal) wavelength extent than the observed spectrum.
- **spec_mod** (*numpy 1d ndarray*) – Model spectrum. It does not require the same wavelength sampling as the observed spectrum. If higher spectral resolution, it will be convolved with the instrumental spectral PSF (if `instru_res` is provided) and then binned to the same sampling. If lower spectral resolution, a linear interpolation is performed to infer the value at the observed spectrum wavelength sampling.
- **dlbda_obs** (*numpy 1d ndarray or list, optional*) – Respective spectral channel width or FWHM of the photometric filters used for each point of the observed spectrum. This vector is used to infer which part(s) of a combined spectro+photometric spectrum should involve convolution+subsampling (model resolution higher than measurements), interpolation (the opposite), or convolution by the transmission curve of a photometric filter. If not provided, it will be inferred from the difference between consecutive lbda_obs points (i.e. inaccurate for a combined spectrum). It must be provided IF one wants to weigh each measurement based on the spectral resolution of each instrument (as in [OLO16]), through the `use_weights` argument.
- **instru_corr** (*numpy 2d ndarray, optional*) – Spectral correlation between post-processed images in which the spectrum is measured. It is specific to the instrument, PSF subtraction algorithm and radial separation of the companion from the central star. Can be computed using `special.spec_corr.spectral_correlation`. In case of a spectrum obtained with different instruments, it is recommended to construct the final spectral_correlation matrix with `special.spec_corr.combine_corrs`. If `instru_corr` is not provided, the uncertainties in each spectral channel will be considered independent. See [GRE16] for more details.
- **instru_res** (*float or list of floats/strings, optional*) – The mean instrumental resolving power(s) OR filter names. This is used to convolve the model spectrum. If several instruments are used, provide a list of resolving power values / filter names, one for each instrument used.
- **instru_idx** (*numpy 1d array, optional*) – 1d array containing an index representing each instrument used to obtain the spectrum, label them from 0 to the number of instruments (n_{ins}). Zero for points that don't correspond to any of the `instru_res` values provided, and i in $[1, n_{ins}]$ for points associated to instru_res[i-1]. This parameter must be provided if the spectrum consists of points obtained with different instruments.

- **use_weights** (*bool, optional*) – For the likelihood calculation, whether to weigh each point of the spectrum based on the spectral resolution or bandwidth of photometric filters used. Weights will be proportional to `dlbda_obs/lbda_obs` if `dlbda_obs` is provided, or set to 1 for all points otherwise.
- **filter_reader** (*python routine, optional*) – External routine that reads a filter file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains transmission values. Important: if not provided, but strings are detected in instru_res, the default file reader will be used. It assumes the following format for the files:
  - first row contains headers (titles of each column)
  - starting from 2nd row: 1st column: wavelength, 2nd col.: transmission
  - Unit of wavelength can be provided in parentheses of first header key name: e.g. "WL(AA)" for angstrom, "wavelength(mu)" for micrometer or "lambda(nm)" for nanometer. Note: only what is in parentheses matters for the units.
- **plot** (*bool, optional*) – Whether to plot the measured spectrum and the model spectrum.
- **outfile** (*string, optional*) – Path+filename for the plot to be saved (won't be saved if not provided).

Returns
chi_sq – Goodness of fit indicator.

Return type
float

15.3 special.config module

Module with configuration parameters, timing functions and multiprocessing utilities (inspired from `VIP`).

`special.config.`**time_fin**(*start_time*)

Return the execution time of a script.

It requires the initialization with the function time_ini().

`special.config.`**time_ini**(*verbose=True*)

Set and print the time at which the script started.

Returns
start_time – Starting time.

Return type
string

`special.config.`**timing**(*start_time*)

Print the execution time of a script.

It requires the initialization with the function time_ini().

15.4 special.fits module

Module with various fits handling functions (same as in VIP)

special.fits.**info_fits**(*fitsfilename*, ***kwargs*)

Print the information about a fits file.

Parameters

- **fitsfilename** (*str*) – Path to the fits file.
- ****kwargs** (*optional*) – Optional arguments to the astropy.io.fits.open() function. E.g. "output_verify" can be set to ignore, in case of non-standard header.

special.fits.**open_fits**(*fitsfilename*, *n=0*, *header=False*, *ignore_missing_end=False*, *precision=<class 'numpy.float32'>*, *return_memmap=False*, *verbose=True*, ***kwargs*)

Load a fits file into a memory as numpy array.

Parameters

- **fitsfilename** (*string or pathlib.Path*) – Name of the fits file or pathlib.Path object
- **n** (*int, optional*) – It chooses which HDU to open. Default is the first one.
- **header** (*bool, optional*) – Whether to return the header along with the data or not.
- **precision** (*numpy dtype, optional*) – Float precision, by default np.float32 or single precision float.
- **ignore_missing_end** (*bool optional*) – Allows to open fits files with a header missing END card.
- **return_memmap** (*bool, optional*) – If True, the function returns the handle to the FITS file opened by mmap. With the hdulist, array data of each HDU to be accessed with mmap, rather than being read into memory all at once. This is particularly useful for working with very large arrays that cannot fit entirely into physical memory.
- **verbose** (*bool, optional*) – If True prints message of completion.
- ****kwargs** (*optional*) – Optional arguments to the astropy.io.fits.open() function. E.g. "output_verify" can be set to ignore, in case of non-standard header.

Returns

- **hdulist** (*hdulist*) – [memmap=True] FITS file n hdulist.
- **data** (*numpy ndarray*) – [memmap=False] Array containing the frames of the fits-cube.
- **header** (*dict*) – [memmap=False, header=True] Dictionary containing the fits header.

special.fits.**write_fits**(*fitsfilename*, *array*, *header=None*, *output_verify='exception'*, *precision=<class 'numpy.float32'>*, *verbose=True*)

Write array and header into FITS file.

If there is a previous file with the same filename then it's replaced.

Parameters

- **fitsfilename** (*string*) – Full path of the fits file to be written.
- **array** (*numpy ndarray*) – Array to be written into a fits file.
- **header** (*numpy ndarray, optional*) – Array with header.

- **output_verify** (*str, optional*) – {"fix", "silentfix", "ignore", "warn", "exception"} Verification options: https://docs.astropy.org/en/stable/io/fits/api/verification.html
- **precision** (*numpy dtype, optional*) – Float precision, by default np.float32 or single precision float.
- **verbose** (*bool, optional*) – If True prints message.

15.5 special.mcmc_sampling module

Module with the MCMC (`emcee`) sampling for model spectra parameter estimation.

`special.mcmc_sampling.`**chain_zero_truncated**(*chain*, *ln_proba=None*, *ar=None*)

Return the Markov chain with the dimension: walkers x steps* x parameters, where steps* is the last step before having 0 (not yet constructed chain).

Parameters

- **chain** (*numpy.array*) – The MCMC chain.
- **ln_proba** (*numpy.array, opt*) – Corresponding ln-probabilities.

Returns

- **out** (*numpy.array*) – The truncated MCMC chain, that is to say, the chain which only contains relevant information.
- **out_ln_proba** (*numpy.array*) – If ln_proba is provided as input, out_ln_proba contains the zero-truncated ln-proba (i.e. matching shape with non-zero samples)

`special.mcmc_sampling.`**confidence**(*isamples*, *labels*, *cfd=68.27*, *bins=100*, *gaussian_fit=False*, *weights=None*, *verbose=True*, *save=False*, *output_dir=''*, *bounds=None*, *priors=None*, ***kwargs*)

Determine the highly probable value for each model parameter, as well as the 1-sigma confidence interval.

Parameters

- **isamples** (*numpy.array*) – The independent samples for each model parameter.
- **labels** (*Tuple of strings*) –

  Tuple of labels in the same order as initial_state:

  – first all parameters related to loaded models (e.g. 'Teff', 'logg')

  – then the planet photometric radius 'R', in Jupiter radius

  – (optionally) the flux of emission lines (labels should match those in the em_lines dictionary), in units of the model spectrum (times mu)

  – (optionally) the optical extinction 'Av', in mag

  – (optionally) the ratio of total to selective optical extinction 'Rv'

  – (optionally) 'Tbb1', 'Rbb1', 'Tbb2', 'Rbb2', etc. for each extra bb contribution.
- **cfd** (*float, optional*) – The confidence level given in percentage.
- **bins** (*int, optional*) – The number of bins used to sample the posterior distributions.
- **gaussian_fit** (*boolean, optional*) – If True, a gaussian fit is performed in order to determine (mu,sigma)

- **weights** (*1d numpy ndarray or None, optional*) – An array of weights for each sample.
- **verbose** (*boolean, optional*) – Display information in the shell.
- **save** (*boolean, optional*) – If "True", a txt file with the results is saved in the output repository.
- **bounds** (*dictionary, opt*) – Only used if a text file is saved summarizing results+bounds+priors. Should be the same bounds as provided to the MCMC.
- **priors** (*dictionary, opt*) – Only used if a text file is saved summarizing results+bounds+priors. Should be the same priors as provided to the MCMC.
- **kwargs** (*optional*) – Additional attributes are passed to the matplotlib hist() method.

Returns
out – A 2 elements tuple with the highly probable solution and the confidence interval.

Return type
tuple

special.mcmc_sampling.**lnlike**(*params*, *labels*, *grid_param_list*, *lbda_obs*, *spec_obs*, *err_obs*, *dist*, *model_grid=None*, *model_reader=None*, *em_lines={}*, *em_grid={}*, *dllbda_obs=None*, *instru_corr=None*, *instru_res=None*, *instru_idx=None*, *use_weights=True*, *filter_reader=None*, *AV_bef_bb=False*, *units_obs='si'*, *units_mod='si'*, *interp_order=1*)

Define the likelihood log-function.

Parameters

- **params** (*tuple*) – Set of models parameters for which the model grid has to be interpolated.
- **labels** (*Tuple of strings*) –

  Tuple of labels in the same order as initial_state:

  - first all parameters related to loaded models (e.g. 'Teff', 'logg')
  - then the planet photometric radius 'R', in Jupiter radius
  - (optionally) the flux of emission lines (labels should match those in the em_lines dictionary), in units of the model spectrum (times mu)
  - (optionally) the optical extinction 'Av', in mag
  - (optionally) the ratio of total to selective optical extinction 'Rv'
  - (optionally) 'Tbb1', 'Rbb1', 'Tbb2', 'Rbb2', etc. for each extra bb contribution.
- **grid_param_list** (*list of 1d numpy arrays/lists*) – Should contain list/numpy 1d arrays with available grid of model parameters (should only contain the sampled parameters, not the models themselves). The models will be loaded with model_reader.
- **lbda_obs** (*numpy 1d ndarray or list*) – Wavelengths of observed spectrum. If several instruments were used, the wavelengths should be ordered per instrument, not necessarily as monotonically increasing wavelength. Hereafter, n_{ch} is the length of lbda_obs.
- **spec_obs** (*numpy 1d ndarray or list*) – Observed spectrum for each value of lbda_obs. Should have a length of n_{ch}.
- **err_obs** (*numpy 1d/2d ndarray or list*) – Uncertainties on the observed spectrum. The array (list) can have either a length of n_{ch}, or a shape of $(2, n_{ch})$ for lower (first column) and upper (second column) uncertainties provided.

- **dist** (*float*) – Distance in parsec, used for flux scaling of the models.
- **model_grid** (*numpy N-d array, optional*) – If provided, should contain the grid of model spectra for each free parameter of the given grid. I.e. for a grid of n_T values of T_{eff} and n_g values of log(g), the numpy array should have a shape of $(n_T, n_g, n_{ch}, 2)$, where the last 2 dimensions correspond to wavelength and fluxes respectively. If provided, `model_grid` takes precedence over `model_name`/ `model_reader`.
- **model_reader** (*python routine, opt*) – External routine that reads a model file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains model values. See example routine in `special.model_resampling.interpolate_model` description.
- **em_lines** (*dictionary, opt*) – Dictionary of emission lines to be added on top of the model spectrum. Each dict entry should be the name of the line, assigned to a tuple of 4 values:

  1. the wavelength (in μ m);
  2. a string indicating whether line intensity is expressed in flux ('F'), luminosity ('L') or log(L/LSun) ("LogL");
  3. the FWHM of the gaussian (or None if to be set automatically);
  4. whether the FWHM is expressed in 'nm', 'mu' or 'km/s'.

  The third and fourth can also be set to None. In that case, the FWHM of the gaussian will automatically be set to the equivalent width of the line, calculated from the flux to be injected and the continuum level (measured in the grid model to which the line is injected).

  Examples:

  ```
  >>> em_lines = {'BrG':(2.1667,'F',None, None)};
  >>> em_lines = {'BrG':(2.1667,'LogL', 100, 'km/s')}
  ```

- **em_grid** (*dictionary pointing to lists, opt*) – Dictionary where each entry corresponds to an emission line and points to a list of values to inject for emission line fluxes. For computation efficiency, interpolation will be performed between the points of this grid during the MCMC sampling. Dictionary entries should match those in `labels` and `em_lines`.

  Examples:

  ```
  >>> BrGmin, BrGmax = -5, 5
  >>> em_grid = {'BrG': np.arange(BrGmin, BrGmax, 20)}
  ```

  ```
  >>> BrGmin, BrGmax = -5, 5
  >>> PaBmin, PaBmax = -2, 7
  >>> em_grid = {'PaB': np.arange(PaBmin, PaBmax, 20),
  >>>            'BrG': np.arange(BrGmin, BrGmax, 20)}
  ```

- **dlbda_obs** (*numpy 1d ndarray or list, optional*) – Respective spectral channel width or FWHM of the photometric filters used for each point of the observed spectrum. This vector is used to infer which part(s) of a combined spectro+photometric spectrum should involve convolution+subsampling (model resolution higher than measurements), interpolation (the opposite), or convolution by the transmission curve of a photometric filter. If not provided, it will be inferred from the difference between consecutive lbda_obs points (i.e. inaccurate for a combined spectrum). It must be provided IF one wants to weigh each measurement based on the spectral resolution of each instrument (as in [OLO16]), through the `use_weights` argument.

- **instru_corr** (*numpy 2d ndarray, optional*) – Spectral correlation between post-processed images in which the spectrum is measured. It is specific to the instrument, PSF subtraction algorithm and radial separation of the companion from the central star. Can be computed using `special.spec_corr.spectral_correlation`. In case of a spectrum obtained with different instruments, it is recommended to construct the final spectral_correlation matrix with `special.spec_corr.combine_corrs`. If `instru_corr` is not provided, the uncertainties in each spectral channel will be considered independent. See [GRE16] for more details.
- **instru_res** (*float or list of floats/strings, optional*) – The mean instrumental resolving power(s) OR filter names. This is used to convolve the model spectrum. If several instruments are used, provide a list of resolving power values / filter names, one for each instrument used.
- **instru_idx** (*numpy 1d array, optional*) – 1d array containing an index representing each instrument used to obtain the spectrum, label them from 0 to the number of instruments (n_{ins}). Zero for points that don't correspond to any of the `instru_res` values provided, and i in $[1, n_{ins}]$ for points associated to instru_res[i-1]. This parameter must be provided if the spectrum consists of points obtained with different instruments.
- **use_weights** (*bool, optional*) – For the likelihood calculation, whether to weigh each point of the spectrum based on the spectral resolution or bandwidth of photometric filters used. Weights will be proportional to `dlbda_obs/lbda_obs` if `dlbda_obs` is provided, or set to 1 for all points otherwise.
- **filter_reader** (*python routine, optional*) – External routine that reads a filter file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains transmission values. Important: if not provided, but strings are detected in instru_res, the default file reader will be used. It assumes the following format for the files:
  - first row contains headers (titles of each column)
  - starting from 2nd row: 1st column: wavelength, 2nd col.: transmission
  - Unit of wavelength can be provided in parentheses of first header key name: e.g. "WL(AA)" for angstrom, "wavelength(mu)" for micrometer or "lambda(nm)" for nanometer. Note: only what is in parentheses matters for the units.
- **AV_bef_bb** (*bool, optional*) – If both extinction and an extra bb component are free parameters, whether to apply extinction before adding the BB component (e.g. extinction mostly from circumplanetary dust) or after the BB component (e.g. mostly insterstellar extinction).
- **units_obs** (*str, opt {'si','cgs','jy'}*) – Units of observed spectrum. 'si' for W/m^2/mu; 'cgs' for ergs/s/cm^2/mu or 'jy' for janskys.
- **units_mod** (*str, opt {'si','cgs','jy'}*) – Units of the model. 'si' for W/m^2/mu; 'cgs' for ergs/s/cm^2/mu or 'jy' for janskys. If different to units_obs, the spectrum units will be converted.
- **interp_order** (*int or tuple of int, optional, {-1,0,1}*) – Interpolation mode for model interpolation. If a tuple of integers, the length should match the number of grid dimensions and will trigger a different interpolation mode for the different parameters.
  - -1: Order 1 spline interpolation in logspace for the parameter
  - 0: nearest neighbour model
  - 1: Order 1 spline interpolation

Returns
out – The log of the likelihood.

Return type
float

special.mcmc_sampling.**lnprob**(*params*, *labels*, *bounds*, *grid_param_list*, *lbda_obs*, *spec_obs*, *err_obs*, *dist*, *model_grid=None*, *model_reader=None*, *em_lines={}*, *em_grid={}*, *dlbda_obs=None*, *instru_corr=None*, *instru_res=None*, *instru_idx=None*, *use_weights=True*, *filter_reader=None*, *AV_bef_bb=False*, *units_obs='si'*, *units_mod='si'*, *interp_order=1*, *priors=None*, *physical=True*)

Define the probability log-function as the sum between the prior and likelihood log-functions.

Parameters

- **params** (*tuple*) – The model parameters.
- **labels** (*Tuple of strings*) –

  Tuple of labels in the same order as initial_state:

  – first all parameters related to loaded models (e.g. 'Teff', 'logg')

  – then the planet photometric radius 'R', in Jupiter radius

  – (optionally) the flux of emission lines (labels should match those in the `em_lines` dictionary), in units of the model spectrum (times mu)

  – (optionally) the optical extinction 'Av', in mag

  – (optionally) the ratio of total to selective optical extinction 'Rv'

  – (optionally) 'Tbb1', 'Rbb1', 'Tbb2', 'Rbb2', etc. for each extra bb contribution.
- **bounds** (*dictionary*) – Each entry should be associated with a tuple corresponding to lower and upper bounds respectively. Bounds should be provided for ALL model parameters, including 'R' (planet photometric radius). 'Av' (optical extinction) is optional. If provided here, Av will also be fitted. All keywords that are neither 'R', 'Av' nor 'M' will be considered model grid parameters.

  Examples:

```
>>> bounds = {'Teff':(1000,2000), 'logg':(3.0,4.5), 'R':(0.1,5)}
>>> bounds = {'Teff':(1000,2000), 'logg':(3.0,4.5), 'R':(0.1,5),
>>>           'Av':(0.,2.5), 'Rv':(1,5)}
>>> bounds = {'Teff':(1000,2000), 'logg':(3.0,4.5), 'R':(0.1,5),
>>>           'Av':(0.,2.5), 'BrG':(1e-17,1e-15)}
>>> bounds = {'Teff':(2000,3000), 'logg':(3.0,4.5), 'R':(1.,5.),
>>>           'Tbb1':(500,1500), 'Rbb1':(0.1,1.)}
```

- **grid_param_list** (*list of 1d numpy arrays/lists*) – Should contain list/numpy 1d arrays with available grid of model parameters (should only contain the sampled parameters, not the models themselves). The models will be loaded with `model_reader`.
- **lbda_obs** (*numpy 1d ndarray or list*) – Wavelengths of observed spectrum. If several instruments were used, the wavelengths should be ordered per instrument, not necessarily as monotonically increasing wavelength. Hereafter, n_{ch} is the length of `lbda_obs`.
- **spec_obs** (*numpy 1d ndarray or list*) – Observed spectrum for each value of `lbda_obs`. Should have a length of n_{ch}.

- **err_obs** (*numpy 1d/2d ndarray or list*) – Uncertainties on the observed spectrum. The array (list) can have either a length of n_{ch}, or a shape of $(2, n_{ch})$ for lower (first column) and upper (second column) uncertainties provided.
- **dist** (*float*) – Distance in parsec, used for flux scaling of the models.
- **model_grid** (*numpy N-d array, optional*) – If provided, should contain the grid of model spectra for each free parameter of the given grid. I.e. for a grid of n_T values of T_{eff} and n_g values of log(g), the numpy array should have a shape of $(n_T, n_g, n_{ch}, 2)$, where the last 2 dimensions correspond to wavelength and fluxes respectively. If provided, `model_grid` takes precedence over `model_name`/ `model_reader`.
- **model_reader** (*python routine, opt*) – External routine that reads a model file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains model values. See example routine in `special.model_resampling.interpolate_model` description.
- **em_lines** (*dictionary, opt*) – Dictionary of emission lines to be added on top of the model spectrum. Each dict entry should be the name of the line, assigned to a tuple of 4 values:

  1. the wavelength (in μ m);
  2. a string indicating whether line intensity is expressed in flux ('F'), luminosity ('L') or log(L/LSun) ("LogL");
  3. the FWHM of the gaussian (or None if to be set automatically);
  4. whether the FWHM is expressed in 'nm', 'mu' or 'km/s'.

  The third and fourth can also be set to None. In that case, the FWHM of the gaussian will automatically be set to the equivalent width of the line, calculated from the flux to be injected and the continuum level (measured in the grid model to which the line is injected).

  Examples:

  ```
  >>> em_lines = {'BrG':(2.1667,'F',None, None)};
  >>> em_lines = {'BrG':(2.1667,'LogL', 100, 'km/s')}
  ```

- **em_grid** (*dictionary pointing to lists, opt*) – Dictionary where each entry corresponds to an emission line and points to a list of values to inject for emission line fluxes. For computation efficiency, interpolation will be performed between the points of this grid during the MCMC sampling. Dictionary entries should match those in `labels` and `em_lines`.

  Examples:

  ```
  >>> BrGmin, BrGmax = -5, 5
  >>> em_grid = {'BrG': np.arange(BrGmin, BrGmax, 20)}
  ```

  ```
  >>> BrGmin, BrGmax = -5, 5
  >>> PaBmin, PaBmax = -2, 7
  >>> em_grid = {'PaB': np.arange(PaBmin, PaBmax, 20),
  >>>            'BrG': np.arange(BrGmin, BrGmax, 20)}
  ```

- **dlbda_obs** (*numpy 1d ndarray or list, optional*) – Respective spectral channel width or FWHM of the photometric filters used for each point of the observed spectrum. This vector is used to infer which part(s) of a combined spectro+photometric spectrum should involve convolution+subsampling (model resolution higher than measurements), interpolation (the opposite), or convolution by the transmission curve of a photometric filter. If not provided, it will be inferred from the difference between consecutive lbda_obs points (i.e.

inaccurate for a combined spectrum). It must be provided IF one wants to weigh each measurement based on the spectral resolution of each instrument (as in [OLO16]), through the `use_weights` argument.

- **instru_corr** (*numpy 2d ndarray, optional*) – Spectral correlation between post-processed images in which the spectrum is measured. It is specific to the instrument, PSF subtraction algorithm and radial separation of the companion from the central star. Can be computed using `special.spec_corr.spectral_correlation`. In case of a spectrum obtained with different instruments, it is recommended to construct the final spectral_correlation matrix with `special.spec_corr.combine_corrs`. If `instru_corr` is not provided, the uncertainties in each spectral channel will be considered independent. See [GRE16] for more details.
- **instru_res** (*float or list of floats/strings, optional*) – The mean instrumental resolving power(s) OR filter names. This is used to convolve the model spectrum. If several instruments are used, provide a list of resolving power values / filter names, one for each instrument used.
- **instru_idx** (*numpy 1d array, optional*) – 1d array containing an index representing each instrument used to obtain the spectrum, label them from 0 to the number of instruments (n_{ins}). Zero for points that don't correspond to any of the `instru_res` values provided, and i in $[1, n_{ins}]$ for points associated to instru_res[i-1]. This parameter must be provided if the spectrum consists of points obtained with different instruments.
- **use_weights** (*bool, optional*) – For the likelihood calculation, whether to weigh each point of the spectrum based on the spectral resolution or bandwidth of photometric filters used. Weights will be proportional to `dlbda_obs/lbda_obs` if `dlbda_obs` is provided, or set to 1 for all points otherwise.
- **filter_reader** (*python routine, optional*) – External routine that reads a filter file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains transmission values. Important: if not provided, but strings are detected in instru_res, the default file reader will be used. It assumes the following format for the files:
  - first row contains headers (titles of each column)
  - starting from 2nd row: 1st column: wavelength, 2nd col.: transmission
  - Unit of wavelength can be provided in parentheses of first header key name: e.g. "WL(AA)" for angstrom, "wavelength(mu)" for micrometer or "lambda(nm)" for nanometer. Note: only what is in parentheses matters for the units.
- **AV_bef_bb** (*bool, optional*) – If both extinction and an extra bb component are free parameters, whether to apply extinction before adding the BB component (e.g. extinction mostly from circumplanetary dust) or after the BB component (e.g. mostly insterstellar extinction).
- **units_obs** (*str, opt {'si','cgs','jy'}*) – Units of observed spectrum. 'si' for W/m^2/mu; 'cgs' for ergs/s/cm^2/mu or 'jy' for janskys.
- **units_mod** (*str, opt {'si','cgs','jy'}*) – Units of the model. 'si' for W/m^2/mu; 'cgs' for ergs/s/cm^2/mu or 'jy' for janskys. If different to units_obs, the spectrum units will be converted.
- **interp_order** (*int, opt, {-1,0,1}*) –

  Interpolation mode for model interpolation.
  - -1: log interpolation (i.e. linear interpolatlion on log(Flux))
  - 0: nearest neighbour model

- 1: Order 1 spline interpolation

- **priors** (*dictionary, opt*) – If not None, sets prior estimates for each parameter of the model. Each entry should be set to either None (no prior) or a tuple of 2 elements containing prior estimate and uncertainty on the estimate. Missing entries (i.e. provided in `bounds` dictionary but not here) will be associated no prior. 'M' can be used for a prior on the mass of the planet. In that case the corresponding prior log probability is computed from the values for parameters 'logg' and 'R'.

  Examples:

  ```
  >>> priors = {'logg':(3.5,0.2), 'Rv':(3.1,0.2)}
  >>> priors = {'Teff':(1600,100), 'logg':(3.5,0.5), 'R':(1.6,0.1),
  >>>           'Av':(1.8,0.2), 'M':(10,3)}
  ```

  Important: dictionary entry names should match exactly those of `bounds`.

- **physical** (*bool, opt*) – In case of extra black body component(s) to a photosphere, whether to force lower temperature than the photosphere effective temperature.

Returns
out – The probability log-function.

Return type
float

special.mcmc_sampling.**mcmc_spec_sampling**(*lbda_obs*, *spec_obs*, *err_obs*, *dist*, *grid_param_list*, *initial_state*, *labels*, *bounds*, *resamp_before=True*, *model_grid=None*, *model_reader=None*, *em_lines={}*, *em_grid={}*, *dlbda_obs=None*, *instru_corr=None*, *instru_res=None*, *instru_idx=None*, *use_weights=True*, *filter_reader=None*, *AV_bef_bb=False*, *units_obs='si'*, *units_mod='si'*, *interp_order=1*, *priors=None*, *physical=True*, *interp_nonexist=True*, *ini_ball=0.1*, *a=2.0*, *nwalkers=1000*, *niteration_min=10*, *niteration_limit=1000*, *niteration_supp=0*, *check_maxgap=20*, *conv_test='ac'*, *ac_c=50*, *ac_count_thr=3*, *burnin=0.3*, *rhat_threshold=1.01*, *rhat_count_threshold=1*, *grid_name='resamp_grid.fits'*, *output_dir='special/'*, *output_file=None*, *nproc=1*, *display=False*, *verbosity=0*, *save=False*)

Runs an affine invariant MCMC sampling algorithm in order to determine the most likely parameters for given spectral model and observed spectrum. The code relies on `emcee` ([FOR13] & [FOR19]).

Allowed features:

- Spectral models can either be read from a grid (e.g. BT-SETTL) or be purely parametric (e.g. a blackbody model).
- Extinction (A_V) and total-to-selective optical extinction ratio (R_V) can be sampled. Default: A_V=0. If non-zero, default R_V=3.1 (ISM).
- A dictionary of emission lines can be provided and their flux can be sampled too.
- Gaussian priors can be provided for each parameter, including the mass of the object. The latter will be used if 'logg' is a parameter.
- Spectral correlation between measurements will be taken into account if provided in 'instru_corr'.
- Convolution of the model spectra with instrumental FWHM or photometric filter can be performed using 'instru_res' and/or 'filter_reader' (done before resampling to observed).

- The weight of each observed point will be directly proportional to Delta lbda_i/lbda_i, where Delta lbda_i is either the FWHM of the photometric filter (imager) or the width of the spectral channel (IFS).
- MCMC convergence criterion can either be based on auto-correlation time (default) or the Gelman-Rubin test.

The result of this procedure is a chain with the samples from the posterior distributions of each of the free parameters in the model.

Parameters

- **lbda_obs** (*numpy 1d ndarray or list*) – Wavelengths of observed spectrum. If several instruments were used, the wavelengths should be ordered per instrument, not necessarily as monotonically increasing wavelength. Hereafter, n_{ch} is the length of `lbda_obs`.
- **spec_obs** (*numpy 1d ndarray or list*) – Observed spectrum for each value of `lbda_obs`. Should have a length of n_{ch}.
- **err_obs** (*numpy 1d/2d ndarray or list*) – Uncertainties on the observed spectrum. The array (list) can have either a length of n_{ch}, or a shape of $(2, n_{ch})$ for lower (first column) and upper (second column) uncertainties provided.
- **dist** (*float*) – Distance in parsec, used for flux scaling of the models.
- **grid_param_list** (*list of 1d numpy arrays/lists*) – Should contain list/numpy 1d arrays with available grid of model parameters (should only contain the sampled parameters, not the models themselves). The models will be loaded with `model_reader`.
- **initial_state** (*tuple of floats*) – Initial guess on the best fit parameters of the spectral fit. Length of the tuple should match the total number of free parameters. Walkers will all be initialised in a small ball of parameter space around that first guess.
  - first all parameters related to loaded models (e.g. 'Teff', 'logg')
  - then the planet photometric radius 'R', in Jupiter radius
  - (optionally) the intensity of emission lines (labels must match those in the em_lines dict), in units of the model spectrum (x mu)
  - (optionally) the optical extinction 'Av', in mag
  - (optionally) the ratio of total to selective optical extinction 'Rv'
  - (optionally) 'Tbb1', 'Rbb1', 'Tbb2', 'Rbb2', etc. for each extra bb contribution
- **labels** (*Tuple of strings*) –

  Tuple of labels in the same order as initial_state:
  - first all parameters related to loaded models (e.g. 'Teff', 'logg')
  - then the planet photometric radius 'R', in Jupiter radius
  - (optionally) the flux of emission lines (labels should match those in the em_lines dictionary), in units of the model spectrum (times mu)
  - (optionally) the optical extinction 'Av', in mag
  - (optionally) the ratio of total to selective optical extinction 'Rv'
  - (optionally) 'Tbb1', 'Rbb1', 'Tbb2', 'Rbb2', etc. for each extra bb contribution.
- **bounds** (*dictionary*) – Each entry should be associated with a tuple corresponding to lower and upper bounds respectively. Bounds should be provided for ALL model parameters, including 'R' (planet photometric radius). 'Av' (optical extinction) is optional. If provided

here, Av will also be fitted. All keywords that are neither 'R', 'Av' nor 'M' will be considered model grid parameters.

Examples:

```
>>> bounds = {'Teff':(1000,2000), 'logg':(3.0,4.5), 'R':(0.1,5)}
>>> bounds = {'Teff':(1000,2000), 'logg':(3.0,4.5), 'R':(0.1,5),
>>>           'Av':(0.,2.5), 'Rv':(1,5)}
>>> bounds = {'Teff':(1000,2000), 'logg':(3.0,4.5), 'R':(0.1,5),
>>>           'Av':(0.,2.5), 'BrG':(1e-17,1e-15)}
>>> bounds = {'Teff':(2000,3000), 'logg':(3.0,4.5), 'R':(1.,5.),
>>>           'Tbb1':(500,1500), 'Rbb1':(0.1,1.)}
```

- **resamp_before** (*bool, optional*) – Whether to prepare the whole grid of resampled models before entering the MCMC, i.e. to avoid doing it at every MCMC step. Recommended. Only reason not to: model grid is too large and individual models require being opened and resampled at each step.
- **model_grid** (*numpy N-d array, optional*) – If provided, should contain the grid of model spectra for each free parameter of the given grid. I.e. for a grid of n_T values of T_{eff} and n_g values of log(g), the numpy array should have a shape of $(n_T, n_g, n_{ch}, 2)$, where the last 2 dimensions correspond to wavelength and fluxes respectively. If provided, `model_grid` takes precedence over `model_name`/ `model_reader`.
- **model_reader** (*python routine, opt*) – External routine that reads a model file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains model values. See example routine in `special.model_resampling.interpolate_model` description.
- **em_lines** (*dictionary, opt*) – Dictionary of emission lines to be added on top of the model spectrum. Each dict entry should be the name of the line, assigned to a tuple of 4 values:

  1. the wavelength (in μ m);
  2. a string indicating whether line intensity is expressed in flux ('F'), luminosity ('L') or log(L/LSun) ("LogL");
  3. the FWHM of the gaussian (or None if to be set automatically);
  4. whether the FWHM is expressed in 'nm', 'mu' or 'km/s'.

  The third and fourth can also be set to None. In that case, the FWHM of the gaussian will automatically be set to the equivalent width of the line, calculated from the flux to be injected and the continuum level (measured in the grid model to which the line is injected).

  Examples:

  ```
  >>> em_lines = {'BrG':(2.1667,'F',None, None)}
  >>> em_lines = {'BrG':(2.1667,'LogL', 100, 'km/s')}
  ```

- **em_grid** (*dictionary pointing to lists, opt*) – Dictionary where each entry corresponds to an emission line and points to a list of values to inject for emission line fluxes. For computation efficiency, interpolation will be performed between the points of this grid during the MCMC sampling. Dictionary entries should match those in `labels` and `em_lines`.

  Examples:

  ```
  >>> BrGmin, BrGmax = -5, 5
  >>> em_grid = {'BrG': np.arange(BrGmin, BrGmax, 20)}
  ```

```
>>> BrGmin, BrGmax = -5, 5
>>> PaBmin, PaBmax = -2, 7
>>> em_grid = {'PaB': np.arange(PaBmin, PaBmax, 20),
>>>            'BrG': np.arange(BrGmin, BrGmax, 20)}
```

- **dlbda_obs** (*numpy 1d ndarray or list, optional*) – Respective spectral channel width or FWHM of the photometric filters used for each point of the observed spectrum. This vector is used to infer which part(s) of a combined spectro+photometric spectrum should involve convolution+subsampling (model resolution higher than measurements), interpolation (the opposite), or convolution by the transmission curve of a photometric filter. If not provided, it will be inferred from the difference between consecutive lbda_obs points (i.e. inaccurate for a combined spectrum). It must be provided IF one wants to weigh each measurement based on the spectral resolution of each instrument (as in [OLO16]), through the `use_weights` argument.
- **instru_corr** (*numpy 2d ndarray, optional*) – Spectral correlation between post-processed images in which the spectrum is measured. It is specific to the instrument, PSF subtraction algorithm and radial separation of the companion from the central star. Can be computed using `special.spec_corr.spectral_correlation`. In case of a spectrum obtained with different instruments, it is recommended to construct the final spectral_correlation matrix with `special.spec_corr.combine_corrs`. If `instru_corr` is not provided, the uncertainties in each spectral channel will be considered independent. See [GRE16] for more details.
- **instru_res** (*float or list of floats/strings, optional*) – The mean instrumental resolving power(s) OR filter names. This is used to convolve the model spectrum. If several instruments are used, provide a list of resolving power values / filter names, one for each instrument used.
- **instru_idx** (*numpy 1d array, optional*) – 1d array containing an index representing each instrument used to obtain the spectrum, label them from 0 to the number of instruments (n_{ins}). Zero for points that don't correspond to any of the `instru_res` values provided, and i in $[1, n_{ins}]$ for points associated to instru_res[i-1]. This parameter must be provided if the spectrum consists of points obtained with different instruments.
- **use_weights** (*bool, optional*) – For the likelihood calculation, whether to weigh each point of the spectrum based on the spectral resolution or bandwidth of photometric filters used. Weights will be proportional to `dlbda_obs/lbda_obs` if `dlbda_obs` is provided, or set to 1 for all points otherwise.
- **filter_reader** (*python routine, optional*) – External routine that reads a filter file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains transmission values. Important: if not provided, but strings are detected in instru_res, the default file reader will be used. It assumes the following format for the files:
  - first row contains headers (titles of each column)
  - starting from 2nd row: 1st column: wavelength, 2nd col.: transmission
  - Unit of wavelength can be provided in parentheses of first header key name: e.g. "WL(AA)" for angstrom, "wavelength(mu)" for micrometer or "lambda(nm)" for nanometer. Note: only what is in parentheses matters for the units.
- **AV_bef_bb** (*bool, optional*) – If both extinction and an extra bb component are free parameters, whether to apply extinction before adding the BB component (e.g. extinction mostly from circumplanetary dust) or after the BB component (e.g. mostly insterstellar extinction).

- **units_obs** (*str, opt {'si','cgs','jy'}*) – Units of observed spectrum. 'si' for W/m^2/mu; 'cgs' for ergs/s/cm^2/mu or 'jy' for janskys.
- **units_mod** (*str, opt {'si','cgs','jy'}*) – Units of the model. 'si' for W/m^2/mu; 'cgs' for ergs/s/cm^2/mu or 'jy' for janskys. If different to units_obs, the spectrum units will be converted.
- **interp_order** (*int, opt, {-1,0,1}*) –

  Interpolation mode for model interpolation.

  - -1: log interpolation (i.e. linear interpolatlion on log(Flux))
  - 0: nearest neighbour model
  - 1: Order 1 spline interpolation
- **priors** (*dictionary, opt*) – If not None, sets prior estimates for each parameter of the model. Each entry should be set to either None (no prior) or a tuple of 2 elements containing prior estimate and uncertainty on the estimate. Missing entries (i.e. provided in `bounds` dictionary but not here) will be associated no prior. 'M' can be used for a prior on the mass of the planet. In that case the corresponding prior log probability is computed from the values for parameters 'logg' and 'R'.

  Examples:

  ```
  >>> priors = {'logg':(3.5,0.2), 'Rv':(3.1,0.2)}
  >>> priors = {'Teff':(1600,100), 'logg':(3.5,0.5), 'R':(1.6,0.1),
  >>>           'Av':(1.8,0.2), 'M':(10,3)}
  ```

  Important: dictionary entry names should match exactly those of `bounds`.
- **physical** (*bool, opt*) – In case of extra black body component(s) to a photosphere, whether to force lower temperature than the photosphere effective temperature.
- **interp_nonexist** (*bool, opt*) – Whether to interpolate non-existing models in the grid. Only used if resamp_before is set to True.
- **ini_ball** (*float or string, default=1e-1*) – Size of the initial ball in parameter space from which walkers start their chain. If "uniform" is provided, a uniform ini_ball spanning the bounds interval will be used to initialise walkers.
- **a** (*float, default=2.0*) – The proposal scale parameter. See notes.
- **nwalkers** (*int, default: 1000*) – Number of walkers
- **niteration_min** (*int, optional*) – Steps per walker lower bound. The simulation will run at least this number of steps per walker.
- **niteration_limit** (*int, optional*) – Steps per walker upper bound. If the simulation runs up to 'niteration_limit' steps without having reached the convergence criterion, the run is stopped.
- **niteration_supp** (*int, optional*) – Number of iterations to run after having "reached the convergence".
- **burnin** (*float, default=0.3*) – The fraction of a walker which is discarded.
- **rhat_threshold** (*float, default=0.01*) – The Gelman-Rubin threshold used for the test for nonconvergence.
- **rhat_count_threshold** (*int, optional*) – The Gelman-Rubin test must be satisfied 'rhat_count_threshold' times in a row before claiming that the chain has converged.

- **check_maxgap** (*int, optional*) – Maximum number of steps per walker between two convergence tests.
- **conv_test** (*str, optional {'gb','autocorr'}*) –

  Method to check for convergence:

  - 'gb' for gelman-rubin test (http://digitalassets.lib.berkeley.edu/sdtr/ucb/text/305.pdf)
  - 'autocorr' for autocorrelation analysis (https://emcee.readthedocs.io/en/stable/tutorials/autocorr/)
- **nproc** (*int, optional*) – The number of processes to use for parallelization.
- **grid_name** (*str, optional*) – Name of the fits file containing the model grid (numpy array) AFTER convolution+resampling as the observed spectrum given as input. If provided, will read it if it exists (and resamp_before is set to True), or make it and write it if it doesn't.
- **output_dir** (*str, optional*) – The name of the output directory which contains the output files in the case `save` is True.
- **output_file** (*str, optional*) – The name of the output file which contains the MCMC results in the case `save` is True.
- **display** (*bool, optional*) – If True, the walk plot is displayed at each evaluation of the Gelman- Rubin test.
- **verbosity** (*0, 1 or 2, optional*) – Verbosity level. 0 for no output and 2 for full information.
- **save** (*bool, optional*) – If True, the MCMC results are pickled.

Returns

- **out** (*numpy.array*) – The MCMC samples after truncation of zeros.
- **lnprobability** (*emcee sample object*) – The corresponding probabilities for each sample

---

Note:

- The parameter *a* must be > 1. For more theoretical information concerning this parameter, see [GOO10].
- The parameter 'rhat_threshold' can be a numpy.array with individual threshold value for each model parameter.
- If several filter filenames are provided in `instru_res`, the filter files must all have the same format and wavelength units (for reading by the same `filter_reader` snippet or default function).
- `grid_param_list` and `model_grid` shouldn't contain grids on radius and Av. For a combined grid model + black body fit, just provide the list of parameters probed by the grid to `grid_param_list`, and provide values for 'Tbbn' and 'Rbbn' to `initial_state`, `labels` and `bounds`.

Examples

This is a minimal example for the file_reader routine to be provided as argument to model_interpolation. The routine should only take as inputs grid parameters, and returns as output: both the wavelengths and model values as a 2D numpy array. This example assumes the model is in a fits file, that is already a 2D numpy array, where the first column is the wavelength, and 2nd column is the corresponding model values.

```
>>> def _example_file_reader(params):
>>>     model = open_fits(filename.format(params[0],params[1]))
>>>     return model
```

special.mcmc_sampling.**show_corner_plot**(*chain*, *burnin=0.5*, *save=False*, *output_dir=''*, *mcmc_res=None*, *units=None*, *ndig=None*, *labels_plot=None*, *plot_name='corner_plot.pdf'*, ***kwargs*)

Display/save a figure showing the corner plot (pdfs + correlation plots).

Parameters

- **chain** (*numpy.array*) – The Markov chain. The shape of chain must be nwalkers x length x dim. If a part of the chain is filled with zero values, the method will discard these steps.
- **burnin** (*float, default: 0*) – The fraction of a walker we want to discard.
- **save** (*boolean, default: False*) – If True, a pdf file is created.
- **output_dir** (*str, optional*) – The name of the output directory which contains the output files in the case `save` is True.
- **mcmc_res** (*numpy array OR tuple of 2 dictionaries/np.array, opt*) – Values to be printed on top of each 1d posterior distribution
  - **if numpy array: - shape $(n_{par}, 3)$, with $(n_{par}$ the number of parameters, with the first, second and third rows containing the most likely value of each parameter and the lower and upper uncertainties at the desired quantiles, respectively.**
    - shape $(n_{par}, 2)$: same as above but with a single value for the uncertainty.
  - if tuple of 2 dictionaries: output of `special.mcm_sampling.spec_confidence` without gaussian fit.
  - if tuple of 2 np.array: output of `special.mcm_sampling.spec_confidence` with gaussian fit.
- **units** (*tuple, opt*) – Tuple of strings containing units for each parameter. If provided, mcmc_res will be printed on top of each 1d posterior distribution along with these units.
- **ndig** (*tuple, opt*) – Number of digits precision for each printed parameter
- **labels_plot** (*tuple, opt*) – Labels corresponding to parameter names, used for the plot. If None, will use `label` passed in kwargs.
- **kwargs** – Additional attributes passed to the `corner.corner` method. (e.g. `labels`, `labels_tit`, `labels_tit_unit`, `title_kwargs`)

Returns

- *Display the figure or create a pdf file named walk_plot.pdf in the working*
- *directory.*

Raises

ImportError –

special.mcmc_sampling.**show_walk_plot**(*chain*, *labels*, *save=False*, *output_dir=''*, *ntrunc=100*, ***kwargs*)

Display/save a figure showing the path of each walker during the MCMC run.

Parameters

- **chain** (*numpy.array*) – The Markov chain. The shape of chain must be nwalkers x length x dim. If a part of the chain is filled with zero values, the method will discard these steps.
- **labels** (*Tuple of strings*) –

  Tuple of labels in the same order as initial_state:

  - first all parameters related to loaded models (e.g. 'Teff', 'logg')
  - then the planet photometric radius 'R', in Jupiter radius
  - (optionally) the flux of emission lines (labels should match those in the em_lines dictionary), in units of the model spectrum (times mu)
  - (optionally) the optical extinction 'Av', in mag
  - (optionally) the ratio of total to selective optical extinction 'Rv'
  - (optionally) 'Tbb1', 'Rbb1', 'Tbb2', 'Rbb2', etc. for each extra bb contribution.
- **save** (*boolean, default: False*) – If True, a pdf file is created.
- **output_dir** (*str, optional*) – The name of the output directory which contains the output files in the case `save` is True.
- **ntrunc** (*int, opt*) – max number of walkers plotted. If too many walkers the plot will become too voluminous and too crowded. Plot will be truncated to only ntrunc first walkers
- **kwargs** – Additional attributes are passed to the matplotlib plot method.

Returns

- *Display the figure or create a pdf file named walk_plot.pdf in the working*
- *directory.*

15.6 special.model_resampling module

Functions useful for spectral fitting of companions, and model interpolation.

special.model_resampling.**interpolate_model**(*params*, *grid_param_list*, *params_em={}*, *em_grid={}*, *em_lines={}*, *labels=None*, *model_grid=None*, *model_reader=None*, *interp_order=1*, *max_dlbda=0.0002*, *verbose=False*)

Parameters

- **params** (*tuple*) – Set of models parameters for which the model grid has to be interpolated.
- **grid_param_list** (*list of 1d numpy arrays/lists*) – Should contain list/numpy 1d arrays with available grid of model parameters (should only contain the sampled parameters, not the models themselves). The models will be loaded with `model_reader`.
- **params_em** (*dictionary, opt*) – Set of emission line parameters (typically fluxes) for which the model grid has to be interpolated.

- **em_grid** (*dictionary pointing to lists, opt*) – Dictionary where each entry corresponds to an emission line and points to a list of values to inject for emission line fluxes. For computation efficiency, interpolation will be performed between the points of this grid during the MCMC sampling. Dictionary entries should match those in `labels` and `em_lines`.

  Examples:

  ```
  >>> BrGmin, BrGmax = -5, 5
  >>> em_grid = {'BrG': np.arange(BrGmin, BrGmax, 20)}
  ```

  ```
  >>> BrGmin, BrGmax = -5, 5
  >>> PaBmin, PaBmax = -2, 7
  >>> em_grid = {'PaB': np.arange(PaBmin, PaBmax, 20),
  >>>            'BrG': np.arange(BrGmin, BrGmax, 20)}
  ```

- **em_lines** (*dictionary, opt*) – Dictionary of emission lines to be added on top of the model spectrum. Each dict entry should be the name of the line, assigned to a tuple of 4 values:

  1. the wavelength (in μ m);
  2. a string indicating whether line intensity is expressed in flux ('F'), luminosity ('L') or log(L/LSun) ("LogL");
  3. the FWHM of the gaussian (or None if to be set automatically);
  4. whether the FWHM is expressed in 'nm', 'mu' or 'km/s'.

  The third and fourth can also be set to None. In that case, the FWHM of the gaussian will automatically be set to the equivalent width of the line, calculated from the flux to be injected and the continuum level (measured in the grid model to which the line is injected).

  Examples:

  ```
  >>> em_lines = {'BrG':(2.1667,'F',None, None)};
  >>> em_lines = {'BrG':(2.1667,'LogL', 100, 'km/s')}
  ```

- **labels** (*Tuple of strings*) –

  Tuple of labels in the same order as initial_state:

  – first all parameters related to loaded models (e.g. 'Teff', 'logg')

  – then the planet photometric radius 'R', in Jupiter radius

  – (optionally) the flux of emission lines (labels should match those in the em_lines dictionary), in units of the model spectrum (times mu)

  – (optionally) the optical extinction 'Av', in mag

  – (optionally) the ratio of total to selective optical extinction 'Rv'

  – (optionally) 'Tbb1', 'Rbb1', 'Tbb2', 'Rbb2', etc. for each extra bb contribution.

  Note: only necessary if an emission line dictionary `em_lines` is provided.

- **model_grid** (*numpy N-d array, optional*) – If provided, should contain the grid of model spectra for each free parameter of the given grid. I.e. for a grid of n_T values of T_{eff} and n_g values of log(g), the numpy array should have a shape of $(n_T, n_g, n_{ch}, 2)$, where the last 2 dimensions correspond to wavelength and fluxes respectively. If provided, `model_grid` takes precedence over `model_name`/ `model_reader`.

- **model_reader** (*python routine, opt*) – External routine that reads a model file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains model values. See example routine in `special.model_resampling.interpolate_model` description.
- **interp_order** (*int, opt, {-1,0,1}*) –

  Interpolation mode for model interpolation.

  - -1: log interpolation (i.e. linear interpolatlion on log(Flux))
  - 0: nearest neighbour model
  - 1: Order 1 spline interpolation
- **max_dlbda** (*float, opt*) – Maximum delta lbda in mu allowed if binning of lbda_model is necessary. This is necessary for grids of models (e.g. BT-SETTL) where the wavelength sampling is not the same depending on parameters (e.g. between 4000K and 4100K models for BT-SETTL): resampling preserving original resolution is too prohibitive computationally.
- **verbose** (*bool, optional*) – Whether to print more information during resampling.

Returns

model – Interpolated model for input parameters. First column corresponds to wavelengths, and the second contains model values.

Return type

2d numpy array

`special.model_resampling.`**make_model_from_params**(*params*, *labels*, *grid_param_list*, *dist*, *lbda_obs=None*, *model_grid=None*, *model_reader=None*, *em_lines={}*, *em_grid={}*, *dlbda_obs=None*, *instru_res=None*, *instru_idx=None*, *filter_reader=None*, *AV_bef_bb=False*, *units_obs='si'*, *units_mod='si'*, *interp_order=1*)

Routine to make the model from input parameters.

Parameters

- **params** (*tuple*) – Set of models parameters for which the model grid has to be interpolated.
- **labels** (*Tuple of strings*) –

  Tuple of labels in the same order as initial_state:

  - first all parameters related to loaded models (e.g. 'Teff', 'logg')
  - then the planet photometric radius 'R', in Jupiter radius
  - (optionally) the flux of emission lines (labels should match those in the em_lines dictionary), in units of the model spectrum (times mu)
  - (optionally) the optical extinction 'Av', in mag
  - (optionally) the ratio of total to selective optical extinction 'Rv'
  - (optionally) 'Tbb1', 'Rbb1', 'Tbb2', 'Rbb2', etc. for each extra bb contribution.
- **grid_param_list** (*list of 1d numpy arrays/lists*) – Should contain list/numpy 1d arrays with available grid of model parameters (should only contain the sampled parameters, not the models themselves). The models will be loaded with `model_reader`.
- **dist** (*float*) – Distance in parsec, used for flux scaling of the models.

- **lbda_obs** (*numpy 1d ndarray or list*) – Wavelengths of observed spectrum. If several instruments were used, the wavelengths should be ordered per instrument, not necessarily as monotonically increasing wavelength. Hereafter, n_{ch} is the length of `lbda_obs`.
- **model_grid** (*numpy N-d array, optional*) – If provided, should contain the grid of model spectra for each free parameter of the given grid. I.e. for a grid of n_T values of T_{eff} and n_g values of $\log(g)$, the numpy array should have a shape of $(n_T, n_g, n_{ch}, 2)$, where the last 2 dimensions correspond to wavelength and fluxes respectively. If provided, `model_grid` takes precedence over `model_name`/ `model_reader`.
- **model_reader** (*python routine, opt*) – External routine that reads a model file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains model values. See example routine in `special.model_resampling.interpolate_model` description.
- **em_lines** (*dictionary, opt*) – Dictionary of emission lines to be added on top of the model spectrum. Each dict entry should be the name of the line, assigned to a tuple of 4 values:

  1. the wavelength (in μ m);
  2. a string indicating whether line intensity is expressed in flux ('F'), luminosity ('L') or log(L/LSun) ("LogL");
  3. the FWHM of the gaussian (or None if to be set automatically);
  4. whether the FWHM is expressed in 'nm', 'mu' or 'km/s'.

  The third and fourth can also be set to None. In that case, the FWHM of the gaussian will automatically be set to the equivalent width of the line, calculated from the flux to be injected and the continuum level (measured in the grid model to which the line is injected).

  Examples:

  ```
  >>> em_lines = {'BrG':(2.1667,'F',None, None)};
  >>> em_lines = {'BrG':(2.1667,'LogL', 100, 'km/s')}
  ```

- **em_grid** (*dictionary pointing to lists, opt*) – Dictionary where each entry corresponds to an emission line and points to a list of values to inject for emission line fluxes. For computation efficiency, interpolation will be performed between the points of this grid during the MCMC sampling. Dictionary entries should match those in `labels` and `em_lines`.

  Examples:

  ```
  >>> BrGmin, BrGmax = -5, 5
  >>> em_grid = {'BrG': np.arange(BrGmin, BrGmax, 20)}
  ```

  ```
  >>> BrGmin, BrGmax = -5, 5
  >>> PaBmin, PaBmax = -2, 7
  >>> em_grid = {'PaB': np.arange(PaBmin, PaBmax, 20),
  >>>            'BrG': np.arange(BrGmin, BrGmax, 20)}
  ```

- **dlbda_obs** (*numpy 1d ndarray or list, optional*) – Respective spectral channel width or FWHM of the photometric filters used for each point of the observed spectrum. This vector is used to infer which part(s) of a combined spectro+photometric spectrum should involve convolution+subsampling (model resolution higher than measurements), interpolation (the opposite), or convolution by the transmission curve of a photometric filter. If not provided, it will be inferred from the difference between consecutive lbda_obs

points (i.e. inaccurate for a combined spectrum). It must be provided IF one wants to weigh each measurement based on the spectral resolution of each instrument (as in [OLO16]), through the `use_weights` argument.

- **instru_res** (*float or list of floats/strings, optional*) – The mean instrumental resolving power(s) OR filter names. This is used to convolve the model spectrum. If several instruments are used, provide a list of resolving power values / filter names, one for each instrument used.
- **instru_idx** (*numpy 1d array, optional*) – 1d array containing an index representing each instrument used to obtain the spectrum, label them from 0 to the number of instruments (n_{ins}). Zero for points that don't correspond to any of the `instru_res` values provided, and i in $[1, n_{ins}]$ for points associated to instru_res[i-1]. This parameter must be provided if the spectrum consists of points obtained with different instruments.
- **filter_reader** (*python routine, optional*) – External routine that reads a filter file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains transmission values. Important: if not provided, but strings are detected in instru_res, the default file reader will be used. It assumes the following format for the files:
  - first row contains headers (titles of each column)
  - starting from 2nd row: 1st column: wavelength, 2nd col.: transmission
  - Unit of wavelength can be provided in parentheses of first header key name: e.g. "WL(AA)" for angstrom, "wavelength(mu)" for micrometer or "lambda(nm)" for nanometer. Note: only what is in parentheses matters for the units.
- **AV_bef_bb** (*bool, optional*) – If both extinction and an extra bb component are free parameters, whether to apply extinction before adding the BB component (e.g. extinction mostly from circumplanetary dust) or after the BB component (e.g. mostly insterstellar extinction).
- **units_obs** (*str, opt {'si','cgs','jy'}*) – Units of observed spectrum. 'si' for W/m^2/mu; 'cgs' for ergs/s/cm^2/mu or 'jy' for janskys.
- **units_mod** (*str, opt {'si','cgs','jy'}*) – Units of the model. 'si' for W/m^2/mu; 'cgs' for ergs/s/cm^2/mu or 'jy' for janskys. If different to units_obs, the spectrum units will be converted.
- **interp_order** (*int or tuple of int, optional, {-1,0,1}*) – Interpolation mode for model interpolation. If a tuple of integers, the length should match the number of grid dimensions and will trigger a different interpolation mode for the different parameters.
  - -1: Order 1 spline interpolation in logspace for the parameter
  - 0: nearest neighbour model
  - 1: Order 1 spline interpolation

Returns

out – The model wavelength and spectrum

Return type

numpy array

Note: `grid_param_list` and `model_grid` shouldn't contain grids on radius and Av. For a combined grid model + black body fit, just provide the list of parameters probed by the grid to `grid_param_list`, and provide

values for 'Tbbn' and 'Rbbn' to `initial_state`, `labels` and `bounds`.

---

special.model_resampling.**make_resampled_models**(*lbda_obs*, *grid_param_list*, *model_grid=None*, *model_reader=None*, *em_lines={}*, *em_grid=None*, *dlbda_obs=None*, *instru_res=None*, *instru_idx=None*, *filter_reader=None*, *interp_nonexist=True*)

Returns a cube of models after convolution and resampling as in the observations.

Parameters

- **lbda_obs** (*numpy 1d ndarray or list*) – Wavelengths of observed spectrum. If several instruments were used, the wavelengths should be ordered per instrument, not necessarily as monotonically increasing wavelength. Hereafter, n_{ch} is the length of `lbda_obs`.
- **grid_param_list** (*list of 1d numpy arrays/lists*) – Should contain list/numpy 1d arrays with available grid of model parameters (should only contain the sampled parameters, not the models themselves). The models will be loaded with `model_reader`.
- **model_grid** (*numpy N-d array, optional*) – If provided, should contain the grid of model spectra for each free parameter of the given grid. I.e. for a grid of n_T values of T_{eff} and n_g values of $\log(g)$, the numpy array should have a shape of $(n_T, n_g, n_{ch}, 2)$, where the last 2 dimensions correspond to wavelength and fluxes respectively. If provided, `model_grid` takes precedence over `model_name`/ `model_reader`.
- **model_reader** (*python routine*) – External routine that reads a model file, converts it to required units and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains model values. Example below.
- **em_lines** (*dictionary, opt*) – Dictionary of emission lines to be added on top of the model spectrum. Each dict entry should be the name of the line, assigned to a tuple of 4 values:

  1. the wavelength (in μ m);
  2. a string indicating whether line intensity is expressed in flux ('F'), luminosity ('L') or log(L/LSun) ("LogL");
  3. the FWHM of the gaussian (or None if to be set automatically);
  4. whether the FWHM is expressed in 'nm', 'mu' or 'km/s'.

  The third and fourth can also be set to None. In that case, the FWHM of the gaussian will automatically be set to the equivalent width of the line, calculated from the flux to be injected and the continuum level (measured in the grid model to which the line is injected).

  Examples:

  ```
  >>> em_lines = {'BrG':(2.1667,'F',None, None)};
  >>> em_lines = {'BrG':(2.1667,'LogL', 100, 'km/s')}
  ```

- **em_grid** (*dictionary pointing to lists, opt*) – Dictionary where each entry corresponds to an emission line and points to a list of values to inject for emission line fluxes. For computation efficiency, interpolation will be performed between the points of this grid during the MCMC sampling. Dictionary entries should match those in `labels` and `em_lines`.

  Examples:

  ```
  >>> BrGmin, BrGmax = -5, 5
  >>> em_grid = {'BrG': np.arange(BrGmin, BrGmax, 20)}
  ```

```
>>> BrGmin, BrGmax = -5, 5
>>> PaBmin, PaBmax = -2, 7
>>> em_grid = {'PaB': np.arange(PaBmin, PaBmax, 20),
>>>            'BrG': np.arange(BrGmin, BrGmax, 20)}
```

- **lbda_mod** (*numpy 1d ndarray or list*) – Wavelength of tested model. Should have a wider wavelength extent than the observed spectrum.
- **spec_mod** (*numpy 1d ndarray*) – Model spectrum. It does not require the same wavelength sampling as the observed spectrum. If higher spectral resolution, it will be convolved with the instrumental spectral PSF (if `instru_res` is provided) and then binned to the same sampling. If lower spectral resolution, a linear interpolation is performed to infer the value at the observed spectrum wavelength sampling.
- **dlbda_obs** (*numpy 1d ndarray or list, optional*) – Respective spectral channel width or FWHM of the photometric filters used for each point of the observed spectrum. This vector is used to infer which part(s) of a combined spectro+photometric spectrum should involve convolution+subsampling (model resolution higher than measurements), interpolation (the opposite), or convolution by the transmission curve of a photometric filter. If not provided, it will be inferred from the difference between consecutive lbda_obs points (i.e. inaccurate for a combined spectrum). It must be provided IF one wants to weigh each measurement based on the spectral resolution of each instrument (as in [OLO16]), through the `use_weights` argument.
- **instru_res** (*float or list of floats/strings, optional*) – The mean instrumental resolving power(s) OR filter names. This is used to convolve the model spectrum. If several instruments are used, provide a list of resolving power values / filter names, one for each instrument used.
- **instru_idx** (*numpy 1d array, optional*) – 1d array containing an index representing each instrument used to obtain the spectrum, label them from 0 to the number of instruments (n_{ins}). Zero for points that don't correspond to any of the `instru_res` values provided, and i in $[1, n_{ins}]$ for points associated to instru_res[i-1]. This parameter must be provided if the spectrum consists of points obtained with different instruments.
- **filter_reader** (*python routine, optional*) – External routine that reads a filter file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains transmission values. Important: if not provided, but strings are detected in instru_res, the default file reader will be used. It assumes the following format for the files:
  - first row contains headers (titles of each column)
  - starting from 2nd row: 1st column: wavelength, 2nd col.: transmission
  - Unit of wavelength can be provided in parentheses of first header key name: e.g. "WL(AA)" for angstrom, "wavelength(mu)" for micrometer or "lambda(nm)" for nanometer. Note: only what is in parentheses matters for the units.
- **interp_nonexist** (*bool, opt*) – Whether to interpolate if models do not exist, based on closest model(s)

Returns

resamp_mod – Grid of model spectra resampled at wavelengths matching the observed spectrum.

Return type

1d numpy array

Note: grid_param_list and model_grid shouldn't contain grids on radius and Av. For a combined grid model + black body fit, just provide the list of parameters probed by the grid to grid_param_list, and provide values for 'Tbbn' and 'Rbbn' to initial_state, labels and bounds.

special.model_resampling.**resample_model**(*lbda_obs*, *lbda_mod*, *spec_mod*, *dlbda_obs=None*, *instru_res=None*, *instru_idx=None*, *filter_reader=None*, *no_constraint=False*, *verbose=False*)

Convolve or interpolate, and resample, a model spectrum to match observed spectrum.

Parameters

- **lbda_obs** (*numpy 1d ndarray or list*) – Wavelengths of observed spectrum. If several instruments were used, the wavelengths should be ordered per instrument, not necessarily as monotonically increasing wavelength. Hereafter, n_{ch} is the length of lbda_obs.
- **lbda_mod** (*numpy 1d ndarray or list*) – Wavelength of tested model. Should have a wider wavelength extent than the observed spectrum.
- **spec_mod** (*numpy 1d ndarray*) – Model spectrum. It does not require the same wavelength sampling as the observed spectrum. If higher spectral resolution, it will be convolved with the instrumental spectral PSF (if instru_res is provided) and then binned to the same sampling. If lower spectral resolution, a linear interpolation is performed to infer the value at the observed spectrum wavelength sampling.
- **dlbda_obs** (*numpy 1d ndarray or list, optional*) – Respective spectral channel width or FWHM of the photometric filters used for each point of the observed spectrum. This vector is used to infer which part(s) of a combined spectro+photometric spectrum should involve convolution+subsampling (model resolution higher than measurements), interpolation (the opposite), or convolution by the transmission curve of a photometric filter. If not provided, it will be inferred from the difference between consecutive lbda_obs points (i.e. inaccurate for a combined spectrum obtained with different instruments).
- **instru_res** (*float or list of floats/strings, optional*) – The mean instrumental resolving power(s) OR filter names. This is used to convolve the model spectrum. If several instruments/resolving powere are to be considered, provide a list of resolving power values or filter names.
- **instru_idx** (*numpy 1d array, optional*) – 1d array containing an index representing each instrument/resolving power used to obtain the spectrum. Label them from 0 to the number of instruments (n_{ins}): zero for points that don't correspond to any of the instru_res values provided, and i in $[1, n_{ins}]$ for points associated to instru_res[i-1]. This parameter must be provided if the spectrum consists of points obtained with different instruments/resolving powers.
- **filter_reader** (*python routine, optional*) – External routine that reads a filter file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains transmission values. Must be provided if instru_res contains strings (filter filenames). Important: if not provided, but strings are detected in instru_res, the default file reader will be used. It assumes the following format for the files:
  - first row contains headers (titles of each column)
  - starting from 2nd row: 1st column: wavelength, 2nd col.: transmission
  - Unit of wavelength can be provided in parentheses of first header key name: e.g. "WL(AA)" for angstrom, "wavelength(mu)" for micrometer or "lambda(nm)" for nanometer. Note: only what is in parentheses matters for the units.

- **no_constraint** (*bool, optional*) – If set to True, will not use 'floor' and 'ceil' constraints when cropping the model wavelength ranges, i.e. faces the risk of extrapolation. May be useful, if the bounds of the wavelength ranges are known to match exactly.
- **verbose** (*bool, optional*) – Whether to print more information during resampling.

Returns

lbda_obs, spec_mod_res – Observed wavelengths, and resampled model spectrum at those wavelengths.

Return type

2d numpy array

15.7 special.nested_sampling module

Module with functions for posterior sampling of the model spectra parameters using nested sampling (either `nestle` or `ultranest` samplers).

special.nested_sampling.**nested_spec_sampling**(*init*, *lbda_obs*, *spec_obs*, *err_obs*, *dist*, *grid_param_list*, *labels*, *bounds*, *resamp_before=True*, *model_grid=None*, *model_reader=None*, *em_lines={}*, *em_grid={}*, *dllbda_obs=None*, *instru_corr=None*, *instru_res=None*, *instru_idx=None*, *use_weights=True*, *filter_reader=None*, *AV_bef_bb=False*, *units_obs='si'*, *units_mod='si'*, *interp_order=1*, *priors=None*, *physical=True*, *interp_nonexist=True*, *output_dir='special/'*, *grid_name='resamp_grid.fits'*, *sampler='ultranest'*, *method='single'*, *npoints=100*, *dlogz=0.2*, *verbose=True*, ***kwargs*)

Runs a nested sampling algorithm in order to retrieve the best-fit parameters for given spectral model and observed spectrum.. The result of this procedure is either a `nestle` [nes13] or `ultranest` [BUC21] object (depending on the chosen sampler) containing the samples from the posterior distributions of each of the parameters. For the `nestle` sampler, several methods are available corresponding to MCMC [SKI04], single ellipsoid [MUK06] or multiple ellipsoid [FER09].

Parameters

- **init** (*numpy ndarray or tuple*) – Initial guess on the best fit parameters of the spectral fit. Length of the tuple should match the total number of free parameters. - first all parameters related to loaded models (e.g. 'Teff', 'logg') - then the planet photometric radius 'R', in Jupiter radius - (optionally) the intensity of emission lines (labels must match those in the em_lines dict), in units of the model spectrum (x mu) - (optionally) the optical extinction 'Av', in mag - (optionally) the ratio of total to selective optical extinction 'Rv' - (optionally) 'Tbb1', 'Rbb1', 'Tbb2', 'Rbb2', etc. for each extra bb contribution
- **lbda_obs** (*numpy 1d ndarray or list*) – Wavelengths of observed spectrum. If several instruments were used, the wavelengths should be ordered per instrument, not necessarily as monotonically increasing wavelength. Hereafter, n_{ch} is the length of `lbda_obs`.
- **spec_obs** (*numpy 1d ndarray or list*) – Observed spectrum for each value of `lbda_obs`. Should have a length of n_{ch}.
- **err_obs** (*numpy 1d/2d ndarray or list*) – Uncertainties on the observed spectrum. The array (list) can have either a length of n_{ch}, or a shape of $(2, n_{ch})$ for lower (first column) and upper (second column) uncertainties provided.
- **dist** (*float*) – Distance in parsec, used for flux scaling of the models.

- **grid_param_list** (*list of 1d numpy arrays/lists*) – Should contain list/numpy 1d arrays with available grid of model parameters (should only contain the sampled parameters, not the models themselves). The models will be loaded with `model_reader`.
- **labels** (*Tuple of strings*) –

  Tuple of labels in the same order as initial_state:

  - first all parameters related to loaded models (e.g. 'Teff', 'logg')
  - then the planet photometric radius 'R', in Jupiter radius
  - (optionally) the flux of emission lines (labels should match those in the em_lines dictionary), in units of the model spectrum (times mu)
  - (optionally) the optical extinction 'Av', in mag
  - (optionally) the ratio of total to selective optical extinction 'Rv'
  - (optionally) 'Tbb1', 'Rbb1', 'Tbb2', 'Rbb2', etc. for each extra bb contribution.
- **bounds** (*dictionary*) – Each entry should be associated with a tuple corresponding to lower and upper bounds respectively. Bounds should be provided for ALL model parameters, including 'R' (planet photometric radius). 'Av' (optical extinction) is optional. If provided here, Av will also be fitted. All keywords that are neither 'R', 'Av' nor 'M' will be considered model grid parameters.

  Examples:

```
>>> bounds = {'Teff':(1000,2000), 'logg':(3.0,4.5), 'R':(0.1,5)}
>>> bounds = {'Teff':(1000,2000), 'logg':(3.0,4.5), 'R':(0.1,5),
>>>           'Av':(0.,2.5), 'Rv':(1,5)}
>>> bounds = {'Teff':(1000,2000), 'logg':(3.0,4.5), 'R':(0.1,5),
>>>           'Av':(0.,2.5), 'BrG':(1e-17,1e-15)}
>>> bounds = {'Teff':(2000,3000), 'logg':(3.0,4.5), 'R':(1.,5.),
>>>           'Tbb1':(500,1500), 'Rbb1':(0.1,1.)}
```

- **resamp_before** (*bool, optional*) – Whether to prepare the whole grid of resampled models before entering the MCMC, i.e. to avoid doing it at every MCMC step. Recommended. Only reason not to: model grid is too large and individual models require being opened and resampled at each step.
- **model_grid** (*numpy N-d array, optional*) – If provided, should contain the grid of model spectra for each free parameter of the given grid. I.e. for a grid of n_T values of T_{eff} and n_g values of log(g), the numpy array should have a shape of $(n_T, n_g, n_{ch}, 2)$, where the last 2 dimensions correspond to wavelength and fluxes respectively. If provided, `model_grid` takes precedence over `model_name`/ `model_reader`.
- **model_reader** (*python routine, opt*) – External routine that reads a model file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains model values. See example routine in `special.model_resampling.interpolate_model` description.
- **em_lines** (*dictionary, opt*) – Dictionary of emission lines to be added on top of the model spectrum. Each dict entry should be the name of the line, assigned to a tuple of 4 values:

  1. the wavelength (in μ m);
  2. a string indicating whether line intensity is expressed in flux ('F'), luminosity ('L') or log(L/LSun) ("LogL");

3. the FWHM of the gaussian (or None if to be set automatically);
4. whether the FWHM is expressed in 'nm', 'mu' or 'km/s'.

The third and fourth can also be set to None. In that case, the FWHM of the gaussian will automatically be set to the equivalent width of the line, calculated from the flux to be injected and the continuum level (measured in the grid model to which the line is injected).

Examples:

```
>>> em_lines = {'BrG':(2.1667,'F',None, None)};
>>> em_lines = {'BrG':(2.1667,'LogL', 100, 'km/s')}
```

- **em_grid** (*dictionary pointing to lists, opt*) – Dictionary where each entry corresponds to an emission line and points to a list of values to inject for emission line fluxes. For computation efficiency, interpolation will be performed between the points of this grid during the MCMC sampling. Dictionary entries should match those in `labels` and `em_lines`.

  Examples:

  ```
  >>> BrGmin, BrGmax = -5, 5
  >>> em_grid = {'BrG': np.arange(BrGmin, BrGmax, 20)}
  ```

  ```
  >>> BrGmin, BrGmax = -5, 5
  >>> PaBmin, PaBmax = -2, 7
  >>> em_grid = {'PaB': np.arange(PaBmin, PaBmax, 20),
  >>>            'BrG': np.arange(BrGmin, BrGmax, 20)}
  ```

- **dlbda_obs** (*numpy 1d ndarray or list, optional*) – Respective spectral channel width or FWHM of the photometric filters used for each point of the observed spectrum. This vector is used to infer which part(s) of a combined spectro+photometric spectrum should involve convolution+subsampling (model resolution higher than measurements), interpolation (the opposite), or convolution by the transmission curve of a photometric filter. If not provided, it will be inferred from the difference between consecutive lbda_obs points (i.e. inaccurate for a combined spectrum). It must be provided IF one wants to weigh each measurement based on the spectral resolution of each instrument (as in [OLO16]), through the `use_weights` argument.

- **instru_corr** (*numpy 2d ndarray, optional*) – Spectral correlation between post-processed images in which the spectrum is measured. It is specific to the instrument, PSF subtraction algorithm and radial separation of the companion from the central star. Can be computed using `special.spec_corr.spectral_correlation`. In case of a spectrum obtained with different instruments, it is recommended to construct the final spectral_correlation matrix with `special.spec_corr.combine_corrs`. If `instru_corr` is not provided, the uncertainties in each spectral channel will be considered independent. See [GRE16] for more details.

- **instru_res** (*float or list of floats/strings, optional*) – The mean instrumental resolving power(s) OR filter names. This is used to convolve the model spectrum. If several instruments are used, provide a list of resolving power values / filter names, one for each instrument used.

- **instru_idx** (*numpy 1d array, optional*) – 1d array containing an index representing each instrument used to obtain the spectrum, label them from 0 to the number of instruments (n_{ins}). Zero for points that don't correspond to any of the `instru_res` values provided, and i in $[1, n_{ins}]$ for points associated to instru_res[i-1]. This parameter must be provided if the spectrum consists of points obtained with different instruments.

- **use_weights** (*bool, optional*) – For the likelihood calculation, whether to weigh each point of the spectrum based on the spectral resolution or bandwidth of photometric filters used. Weights will be proportional to `dlbda_obs/lbda_obs` if `dlbda_obs` is provided, or set to 1 for all points otherwise.
- **filter_reader** (*python routine, optional*) – External routine that reads a filter file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains transmission values. Important: if not provided, but strings are detected in instru_res, the default file reader will be used. It assumes the following format for the files:
  - first row contains headers (titles of each column)
  - starting from 2nd row: 1st column: wavelength, 2nd col.: transmission
  - Unit of wavelength can be provided in parentheses of first header key name: e.g. "WL(AA)" for angstrom, "wavelength(mu)" for micrometer or "lambda(nm)" for nanometer. Note: only what is in parentheses matters for the units.
- **AV_bef_bb** (*bool, optional*) – If both extinction and an extra bb component are free parameters, whether to apply extinction before adding the BB component (e.g. extinction mostly from circumplanetary dust) or after the BB component (e.g. mostly insterstellar extinction).
- **units_obs** (*str, opt {'si','cgs','jy'}*) – Units of observed spectrum. 'si' for W/m^2/mu; 'cgs' for ergs/s/cm^2/mu or 'jy' for janskys.
- **units_mod** (*str, opt {'si','cgs','jy'}*) – Units of the model. 'si' for W/m^2/mu; 'cgs' for ergs/s/cm^2/mu or 'jy' for janskys. If different to units_obs, the spectrum units will be converted.
- **interp_order** (*int or tuple of int, optional, {-1,0,1}*) – Interpolation mode for model interpolation. If a tuple of integers, the length should match the number of grid dimensions and will trigger a different interpolation mode for the different parameters.
  - -1: Order 1 spline interpolation in logspace for the parameter
  - 0: nearest neighbour model
  - 1: Order 1 spline interpolation
- **priors** (*dictionary, opt*) – If not None, sets prior estimates for each parameter of the model. Each entry should be set to either None (no prior) or a tuple of 2 elements containing prior estimate and uncertainty on the estimate. Missing entries (i.e. provided in `bounds` dictionary but not here) will be associated no prior. 'M' can be used for a prior on the mass of the planet. In that case the corresponding prior log probability is computed from the values for parameters 'logg' and 'R'.

  Examples:

  ```
  >>> priors = {'logg':(3.5,0.2), 'Rv':(3.1,0.2)}
  >>> priors = {'Teff':(1600,100), 'logg':(3.5,0.5), 'R':(1.6,0.
  ↪1),
  >>>           'Av':(1.8,0.2), 'M':(10,3)}
  ```

  Important: dictionary entry names should match exactly those of `bounds`.
- **physical** (*bool, opt*) – In case of extra black body component(s) to a photosphere, whether to force lower temperature than the photosphere effective temperature.

- **interp_nonexist** (*bool, opt*) – Whether to interpolate non-existing models in the grid. Only used if resamp_before is set to True.
- **w** (*float or tuple*) – The relative size of the bounds (around the initial state `init`) for each parameter. If a float the same relative size is considered for each parameter. E.g. if 0.1, bounds will be set to: (0.9*params[0], 1.1*params[0]), ... (0.9*params[N-1], 1.1*params[N-1]), to True), or make it and write it if it does not.
- **output_dir** (*str, optional*) – The name of the output directory which contains the output files in the case `save` is True.
- **grid_name** (*str, optional*) – Name of the fits file containing the model grid (numpy array) AFTER convolution+resampling as the observed spectrum given as input. If provided, will read it if it exists (and resamp_before is set
- **method** (*{"single", "multi", "classic"}, str optional*) – Flavor of nested sampling.
- **npoints** (*int optional*) – Number of active points. Must be >> ndim+1, otherwise will produce bad results. For UltraNest, this is the minimum number of active points.
- **dlogz** (*float, optional*) – Target evidence uncertainty. Iterations will stop when the estimated contribution of the remaining prior volume to the total evidence falls below this threshold. Explicitly, the stopping criterion is log(z + z_est) - log(z) < dlogz where z is the current evidence from all saved samples, and z_est is the estimated contribution from the remaining volume. This option and decline_factor are mutually exclusive. If neither is specified, the default is dlogz=0.2.
- **decline_factor** (*float, optional*) – If supplied, iteration will stop when the weight (likelihood times prior volume) of newly saved samples has been declining for decline_factor * nsamples consecutive samples. A value of 1.0 seems to work pretty well.
- **rstate** (*random instance, optional*) – RandomState instance. If not given, the global random state of the numpy.random module will be used.
- **kwargs** (*optional*) – Additional optional arguments to either the *nestle.sample* or *ultranest.ReactiveNestedSampler.run* functions.

Returns

res – `Nestle` object with the nested sampling results, including the posterior samples.

Return type

nestle object

Notes

Nested Sampling is a computational approach for integrating posterior probability in order to compare models in Bayesian statistics. It is similar to Markov Chain Monte Carlo (MCMC) in that it generates samples that can be used to estimate the posterior probability distribution. Unlike MCMC, the nature of the sampling also allows one to calculate the integral of the distribution. It also happens to be a pretty good method for robustly finding global maxima.

Nestle documentation:

http://kbarbary.github.io/nestle/

Convergence:

http://kbarbary.github.io/nestle/stopping.html Nested sampling has no well-defined stopping point. As iterations continue, the active points sample a smaller and smaller region of prior space. This can continue indefinitely. Unlike typical MCMC methods, we don't gain any additional precision on the results by letting the algorithm run longer; the precision is determined at the outset by the number of active points.

So, we want to stop iterations as soon as we think the active points are doing a pretty good job sampling the remaining prior volume - once we've converged to the highest-likelihood regions such that the likelihood is relatively flat within the remaining prior volume.

Method:

The trick in nested sampling is to, at each step in the algorithm, efficiently choose a new point in parameter space drawn with uniform probability from the parameter space with likelihood greater than the current likelihood constraint. The different methods all use the current set of active points as an indicator of where the target parameter space lies, but differ in how they select new points from it:

- "classic" is close to the method described in [SKI04].
- "single" [MUK06] determines a single ellipsoid that bounds all active points, enlarges the ellipsoid by a user-settable factor, and selects a new point at random from within the ellipsoid.
- "multiple" [FER09] (Multinest). In cases where the posterior is multi-modal, the single-ellipsoid method can be extremely inefficient. In such situations, there are clusters of active points on separate high-likelihood regions separated by regions of lower likelihood. Bounding all points in a single ellipsoid means that the ellipsoid includes the lower-likelihood regions we wish to avoid sampling from. The solution is to detect these clusters and bound them in separate ellipsoids. For this, we use a recursive process where we perform K-means clustering with K=2. If the resulting two ellipsoids have a significantly lower total volume than the parent ellipsoid (less than half), we accept the split and repeat the clustering and volume test on each of the two subset of points. This process continues recursively. Alternatively, if the total ellipse volume is significantly greater than expected (based on the expected density of points) this indicates that there may be more than two clusters and that K=2 was not an appropriate cluster division. We therefore still try to subdivide the clusters recursively. However, we still only accept the final split into N clusters if the total volume decrease is significant.

Note:

- If several filter filenames are provided in `instru_res`, the filter files must all have the same format and wavelength units (for reading by the same `filter_reader` snippet or default function).
- `grid_param_list` and `model_grid` shouldn't contain grids on radius and Av. For a combined grid model + black body fit, just provide the list of parameters probed by the grid to `grid_param_list`, and provide values for 'Tbbn' and 'Rbbn' to `initial_state`, `labels` and `bounds`.

---

special.nested_sampling.**show_nestle_results**(*ns_object*, *labels*, *method*, *burnin=0.4*, *bins=None*, *cfd=68.27*, *units=None*, *ndig=None*, *labels_plot=None*, *save=False*, *output_dir='/'*, *plot=False*, ***kwargs*)

Show the results obtained with the Nestle sampler: summary, parameters with errors, walk and corner plots. Returns best-fit values and uncertatinties.

Parameters

- **ns_object** (*numpy.array*) – The nestle object returned from *nested_spec_sampling*.
- **labels** (*Tuple of strings*) –

  Tuple of labels in the same order as initial_state:

  - first all parameters related to loaded models (e.g. 'Teff', 'logg')
  - then the planet photometric radius 'R', in Jupiter radius
  - (optionally) the flux of emission lines (labels should match those in the em_lines dictionary), in units of the model spectrum (times mu)
  - (optionally) the optical extinction 'Av', in mag

- (optionally) the ratio of total to selective optical extinction 'Rv'
- (optionally) 'Tbb1', 'Rbb1', 'Tbb2', 'Rbb2', etc. for each extra bb contribution.

- **method** (*{"single", "multi", "classic"}, str optional*) – Flavor of nested sampling.
- **burnin** (*float, default: 0*) – The fraction of a walker we want to discard.
- **bins** (*int, optional*) – The number of bins used to sample the posterior distributions.
- **cfd** (*float, optional*) – The confidence level given in percentage.
- **units** (*tuple, opt*) – Tuple of strings containing units for each parameter. If provided, mcmc_res will be printed on top of each 1d posterior distribution along with these units.
- **ndig** (*tuple, opt*) – Number of digits precision for each printed parameter.
- **labels_plot** (*tuple, opt*) – Labels corresponding to parameter names, used for the plot. If None, will use "labels" passed in kwargs.
- **save** (*boolean, default: False*) – If True, a pdf file is created.
- **output_dir** (*str, optional*) – The name of the output directory which contains the output files in the case `save` is True.
- **plot** (*bool, optional*) – Whether to show the plots (instead of saving them).
- **kwargs** – Additional optional arguments passed to *confidence* (matplotlib optional arguments for histograms).

Returns

final_res – Best-fit parameters and uncertainties (corresponding to 68% confidence interval). Dimensions: nparams x 2.

Return type

numpy ndarray

special.nested_sampling.**show_ultranest_results**(*un_object*, *labels*, *grid_param_list*, *dist*, *bins=None*, *cfd=68.27*, *ndig=None*, *save=False*, *output_dir='/'*, *plot=False*, *col='b'*, *units=None*, *labels_plot=None*, *lbda_obs=None*, *spec_obs=None*, *spec_obs_err=None*, *n_pts=None*, *title=None*, *figsize=(8, 6)*, ***kwargs*)

Shows the results obtained with the Ultranest sampler: summary, parameters with errors, walk and corner plots. Returns best-fit values and uncertatinties.

Parameters

- **un_object** (*object*) – The UltraNest Sampler object returned by nested_spec_sampling.
- **labels** (*Tuple of strings*) –

  Tuple of labels in the same order as initial_state:

  - first all parameters related to loaded models (e.g. 'Teff', 'logg')
  - then the planet photometric radius 'R', in Jupiter radius
  - (optionally) the flux of emission lines (labels should match those in the em_lines dictionary), in units of the model spectrum (times mu)
  - (optionally) the optical extinction 'Av', in mag
  - (optionally) the ratio of total to selective optical extinction 'Rv'
  - (optionally) 'Tbb1', 'Rbb1', 'Tbb2', 'Rbb2', etc. for each extra bb contribution.

- **grid_param_list** (*list of 1d numpy arrays/lists*) – Should contain list/numpy 1d arrays with available grid of model parameters (should only contain the sampled parameters, not the models themselves). The models will be loaded with `model_reader`.
- **dist** (*float*) – Distance in parsec, used for flux scaling of the models.
- **bins** (*int, optional*) – The number of bins used to sample the posterior distributions.
- **cfd** (*float, optional*) – The confidence level given in percentage.
- **ndig** (*tuple, opt*) – Number of digits precision for each printed parameter.
- **save** (*boolean, default: False*) – If True, a pdf file is created.
- **output_dir** (*str, optional*) – The name of the output directory which contains the output files in the case `save` is True.
- **plot** (*bool, optional*) – Whether to show the best-fit model against data.
- **col** (*str, optional*) – Color used for data points.
- **lbda_obs** (*1d ndarrays, optional*) – Must be provided if plot is set to True
- **spec_obs** (*1d ndarrays, optional*) – Must be provided if plot is set to True
- **spec_obs_err** (*1d ndarrays, optional*) – Must be provided if plot is set to True
- **n_pts** (*None or int, optional*) – If None, models will be sampled at the same resolution as measured spectrum for plot. If an integer, corresponds to the number of sampled points (uniform sampling) between the first and last point of the measured spectrum.
- **title** (*str, optional*) – If plot is set to True, title of the plot.
- **kwargs** – Additional optional arguments passed to *nested_spec_sampling* - only used if plot is set to True and model parameters are evaluated.

Returns

final_res – Best-fit parameters and uncertainties (corresponding to 68% confidence interval). Dimensions: nparams x 2.

Return type

numpy ndarray

15.8 special.spec_corr module

Module to estimate the spectral correlation between channels of an IFS datacube.

special.spec_corr.**combine_spec_corrs**(*arr_list*)

Combines the spectral correlation matrices of different instruments into a single square matrix (required for input of spectral fits).

Parameters

arr_list (*list or tuple of numpy ndarrays*) – List/tuple containing the distinct square spectral correlation matrices OR ones (for independent photometric measurements).

Returns

combi_corr – 2d square ndarray representing the combined spectral correlation.

Return type

numpy 2d ndarray

special.spec_corr.**spectral_correlation**(*array*, *awidth=2*, *r_in=1*, *r_out=None*, *pl_xy=None*, *mask_r=4*, *fwhm=4*, *sp_fwhm_guess=3*, *full_output=False*)

Computes the spectral correlation between (post-processed) IFS frames, as a function of radius, implemented as Eq. 7 of [GRE16]. This is a crucial step for an unbiased fit of a measured IFS spectrum to either synthetic or template spectra.

Parameters

- **array** (*numpy ndarray*) – Input cube or 3d array, of dimensions n_ch x n_y x n_x; where n_y and n_x should be odd values (star should be centered on central pixel).
- **awidth** (*int, optional*) – Width in pixels of the concentric annuli used to compute the spectral correlation as a function of radial separation. Greco & Brandt 2017 noted no significant differences for annuli between 1 and 3 pixels width on GPI data.
- **r_in** (*int, optional*) – Innermost radius where the spectral correlation starts to be computed.
- **r_out** (*int, optional*) – Outermost radius where the spectral correlation is computed. If left as None, it will automatically be computed up to the edge of the frame.
- **pl_xy** (*tuple of tuples of 2 floats, optional*) – x,y coordinates of all companions present in the images. If provided, a circle centered on the location of each companion will be masked out for the spectral correlation computation.
- **mask_r** (*float, optional*) – if pl_xy is provided, this should also be provided. Size of the aperture around each companion (in terms of fwhm) that is discarded to not bias the spectral correlation computation.
- **fwhm** (*float, optional*) – if pl_xy is provided, this should also be provided. By default we consider a 2FWHM aperture mask around each companion to not bias the spectral correlation computation.
- **sp_fwhm_guess** (*float, optional*) – Initial guess on the spectral FWHM of all channels.
- **full_output** (*bool, opt*) – Whether to also output the fitted spectral FWHM for each channel, and the vector of radial separation at which each spectral correlation matrix is calculated.

Returns

- **sp_corr** (*numpy ndarray*) – 3d array of spectral correlation, as a function of radius with dimensions: n_rad x n_ch x n_ch, where n_rad = int((r_out-r_in)/2)
- **sp_fwhm** (*numpy ndarray*) – (if full_output is True) 2d array containing the spectral fwhm at each radius, for each spectral channel. Dims: n_rad x n_ch
- **sp_rad** (*numpy ndarray*) – (if full_output is True) 1d array containing the radial separation of each measured spectral correlation matrix. Dims: n_rad

Note: Radii that are skipped will be filled with zeros in the output cube.

15.9 special.spec_indices module

Module with utilities to estimate the spectral type and gravity of an object based on spectral indices.

special.spec_indices.**digit_to_spt**(*idx*, *convention='splat'*)

Converts an integer index into spectral type.

Parameters

- **idx** (*float or int*) – Index value of the spectral type
- **convention** (*str, optional {'splat', 'Allers+07'}*) – Which convention to use to convert digit into spectral type. Convention from splat: K0 = 0, M0=10, L0=20, T0=30, Y9 = 49 Convention from Allers+07: M0 = 0, L0 = 10, ...

Returns

spt – String representing the spectral index

Return type

str

special.spec_indices.**sp_idx_to_gravity**(*idx*, *name='Na-1.1'*)

Provides a qualitative estimate of the gravity/youth based on a gravity-sensitive spectral index. Implemented so far:

- the Na-1.1 index [ALL07]
- the CO-2.3 index [GOR03]

Parameters

- **idx** (*float*) – Value of spectral index
- **name** (*str, optional {'Na-1.1', 'CO-2.3'}*) – The name of the spectral index.

special.spec_indices.**sp_idx_to_spt**(*idx*, *name='H2O-1.5'*, *idx_err=None*, *young=True*)

Estimates a spectral type from a spectral index. Implemented so far:

- the H2O 1.3 mu index [GOR03]
- the H2O 1.5 mu index [ALL07]
- the H2O-2 index [SLE04]

Note on scale of SpT: 0 = M0, 10 = L0.

Parameters

- **idx** (*float*) – Value of spectral index
- **name** (*str, optional {'H2O-1.3', 'H2O-1.5, 'H2O-2'}*) – The name of the spectral index.
- **idx_err** (*float, optional*) – Uncertainty on the spectral index value
- **young** (*bool, opt*) – Whether the object is likely young (only used for 'H2O-1.3' index, which is gravity dependent)

Returns

- **spt** (*float*) – Value of the spectral type
- **spt_err** (*float*) – [if idx_err is provided] Uncertainty on the spectral type.

special.spec_indices.**spectral_idx**(*lbda*, *spec*, *band='H2O-1.5'*, *spec_err=None*, *verbose=False*)

Computes a spectral index. Implemented so far:

- the Na 1.1 mu index [ALL07]
- the H2O 1.3 mu index [GOR03]
- the H2O 1.5 mu index [ALL07]
- the H2O 2 index [SLE04]
- the CO 2.3 index [GOR03].

Parameters

- **lbda** (*numpy ndarray*) – 1d numpy array containing the wavelengths of the spectrum in microns.
- **spec** (*numpy ndarray*) – 1d numpy array containing the measured flux (arbitrary units accepted).
- **band** (*str, optional {'H2O-1.5', 'H2O-1.3', 'H2O-2', 'Na-1.1', 'CO-2.3'}*) – Name of the band where the spectral index is defined (spectral feature + wavelength in mu)
- **spec_err** (*numpy ndarray, optional*) – 1d numpy array containing the uncertainties on the measured flux (arbitrary units accepted). If provided the uncertainty on the spectral index will also be returned.
- **verbose** (*bool, optional*) – Whther to print more information.

Returns

- **index** (*float*) – Value of the spectral index
- **index_err** (*float*) – [if spec_err is provided] Uncertainty on the spectral index.

special.spec_indices.**spt_to_digit**(*spt*, *convention='splat'*)

Converts a string representing spectral type into an integer index.

Parameters

- **spt** (*str*) – String representing the spectral index
- **convention** (*str, optional {'splat', 'Allers+07'}*) – Which convention to use to convert digit into spectral type. Convention from splat: K0 = 0, M0=10, L0=20, T0=30, Y9 = 49 Convention from [ALL07]: M0 = 0, L0 = 10, ...

Returns

idx – Index value of the spectral type

Return type

float or int

15.10 special.template_fit module

Module for simplex or grid search of best fit spectrum in a template library.

special.template_fit.**best_fit_tmp**(*lbda_obs*, *spec_obs*, *err_obs*, *tmp_reader*, *search_mode='simplex'*, *n_best=1*, *lambda_scal=None*, *scale_range=(0.1, 10, 0.01)*, *ext_range=None*, *simplex_options=None*, *dlbda_obs=None*, *instru_corr=None*, *instru_res=None*, *instru_idx=None*, *filter_reader=None*, *lib_dir='tmp_lib/'*, *tmp_endswith='.fits'*, *red_chi2=True*, *remove_nan=False*, *nproc=1*, *verbosity=0*, *force_continue=False*, *min_npts=1*, ***kwargs*)

Finds the best fit template spectrum to a given observed spectrum, within a spectral library. By default, a single free parameter is considered: the scaling factor of the spectrum. A first automatic scaling is performed by comparing the flux of the observed and template spectra at lambda_scal. Then a more refined scaling is performed, either through simplex or grid search (within scale_range). If fit_extinction is set to True, the exctinction is also considered as a free parameter.

Parameters

- **lbda_obs** (*numpy 1d ndarray or list*) – Wavelengths of observed spectrum. If several instruments were used, the wavelengths should be ordered per instrument, not necessarily as monotonically increasing wavelength. Hereafter, n_{ch} is the length of `lbda_obs`.
- **spec_obs** (*numpy 1d ndarray or list*) – Observed spectrum for each value of `lbda_obs`. Should have a length of n_{ch}.
- **err_obs** (*numpy 1d/2d ndarray or list*) – Uncertainties on the observed spectrum. The array (list) can have either a length of n_{ch}, or a shape of $(2, n_{ch})$ for lower (first column) and upper (second column) uncertainties provided.
- **tmp_reader** (*python routine*) – External routine that reads a model file and returns a 3D numpy array, where the first column corresponds to wavelengths, the second contains flux values, and the third the uncertainties on the flux.
- **search_mode** (*str, optional, {'simplex','grid'}*) – How is the best fit template found? Simplex or grid search.
- **n_best** (*int, optional*) – Number of best templates to be returned (default: 1)
- **lambda_scal** (*float, optional*) – Wavelength where a first scaling will be performed between template and observed spectra. If not provided, the middle wavelength of the osberved spectra will be considered.
- **scale_range** (*tuple, opt*) – If grid search, this parameter should be provided as a tuple of 3 floats: lower limit, upper limit and step of the grid search for the scaling factor to be applied AFTER the first rough scaling (i.e. scale_range should always encompass 1).
- **ext_range** (*tuple or None, opt*) –
  - If None: differential extinction is not considered as a free parameter.
  - If a tuple: it should contain 2 floats (for simplex `search_mode`) or 3 floats (for grid search `search_mode`) corresponding to the lower limit, upper limit (and step for the grid search). For the simplex search, the lower and upper limits are used to set a chi square of infinity outside of the range.
- **dlbda_obs** (*numpy 1d ndarray or list, optional*) – Respective spectral channel width or FWHM of the photometric filters used for each point of the observed spectrum. This vector is used to infer which part(s) of a combined spectro+photometric spectrum

should involve convolution+subsampling (model resolution higher than measurements), interpolation (the opposite), or convolution by the transmission curve of a photometric filter. If not provided, it will be inferred from the difference between consecutive lbda_obs points (i.e. inaccurate for a combined spectrum). It must be provided IF one wants to weigh each measurement based on the spectral resolution of each instrument (as in [OLO16]), through the `use_weights` argument.

- **instru_corr** (*numpy 2d ndarray, optional*) – Spectral correlation between post-processed images in which the spectrum is measured. It is specific to the instrument, PSF subtraction algorithm and radial separation of the companion from the central star. Can be computed using `special.spec_corr.spectral_correlation`. In case of a spectrum obtained with different instruments, it is recommended to construct the final spectral_correlation matrix with `special.spec_corr.combine_corrs`. If `instru_corr` is not provided, the uncertainties in each spectral channel will be considered independent. See [GRE16] for more details.
- **instru_res** (*float or list of floats/strings, optional*) – The mean instrumental resolving power(s) OR filter names. This is used to convolve the model spectrum. If several instruments are used, provide a list of resolving power values / filter names, one for each instrument used.
- **instru_idx** (*numpy 1d array, optional*) – 1d array containing an index representing each instrument used to obtain the spectrum, label them from 0 to the number of instruments (n_{ins}). Zero for points that don't correspond to any of the `instru_res` values provided, and i in $[1, n_{ins}]$ for points associated to instru_res[i-1]. This parameter must be provided if the spectrum consists of points obtained with different instruments.
- **filter_reader** (*python routine, optional*) – External routine that reads a filter file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains transmission values. Important: if not provided, but strings are detected in instru_res, the default file reader will be used. It assumes the following format for the files:
  - first row contains headers (titles of each column)
  - starting from 2nd row: 1st column: wavelength, 2nd col.: transmission
  - Unit of wavelength can be provided in parentheses of first header key name: e.g. "WL(AA)" for angstrom, "wavelength(mu)" for micrometer or "lambda(nm)" for nanometer. Note: only what is in parentheses matters for the units.
- **simplex_options** (*dict, optional*) – The scipy.optimize.minimize simplex (Nelder-Mead) options.
- **red_chi2** (*bool, optional*) – Whether to compute the reduced chi square. If False, considers chi^2.
- **remove_nan** (*bool, optional*) – Whether to remove NaN values from template spectrum BEFORE resampling to the wavelength sampling of the observed spectrum. Whether it is set to True or False, a check is made just before chi^2 is calculated (after resampling), and only non-NaN values will be considered.
- **nproc** (*None or int, optional*) – The number of processes to use for parallelization. If set to None, will automatically use half of the available CPUs on the machine.
- **verbosity** (*0, 1 or 2, optional*) – Verbosity level. 0 for no output and 2 for full information.
- **force_continue** (*bool, optional*) – In case of issue with the fit, whether to continue regardless (this may be useful in an uneven spectral library, where some templates have

too few points for the fit to be performed).

- **min_npts** (*int or None, optional*) – Minimum number of (resampled) points to consider a template spectrum valid in the minimization search.
- ****kwargs** (*optional*) – Optional arguments to the scipy.optimize.minimize function

Returns

- **final_tmpname** (*tuple of n_best str*) – Best-fit template filenames
- **final_tmp** (*tuple of n_best 3D numpy array*) – Best-fit template spectra (3D: lbda+spec+spec_err)
- **final_chi** (*1D numpy array of length n_best*) – Best-fit template chi^2
- **final_params** (*2D numpy array (2xn_best)*) – Best-fit parameters (optimal scaling and optical extinction). Note if extinction is not fitted, optimal AV will be set to 0.

special.template_fit.**get_chi**(*lbda_obs*, *spec_obs*, *err_obs*, *tmp_name*, *tmp_reader*, *search_mode='simplex'*, *lambda_scal=None*, *scale_range=(0.1, 10, 0.01)*, *ext_range=None*, *dlbda_obs=None*, *instru_corr=None*, *instru_res=None*, *instru_idx=None*, *use_weights=True*, *filter_reader=None*, *simplex_options=None*, *red_chi2=True*, *remove_nan=False*, *force_continue=False*, *min_npts=1*, *verbose=False*, ***kwargs*)

Routine calculating chi^2, optimal scaling factor and optimal extinction for a given template spectrum to match an observed spectrum.

Parameters

- **lbda_obs** (*numpy 1d ndarray or list*) – Wavelengths of observed spectrum. If several instruments were used, the wavelengths should be ordered per instrument, not necessarily as monotonically increasing wavelength. Hereafter, n_{ch} is the length of `lbda_obs`.
- **spec_obs** (*numpy 1d ndarray or list*) – Observed spectrum for each value of `lbda_obs`. Should have a length of n_{ch}.
- **err_obs** (*numpy 1d/2d ndarray or list*) – Uncertainties on the observed spectrum. The array (list) can have either a length of n_{ch}, or a shape of $(2, n_{ch})$ for lower (first column) and upper (second column) uncertainties provided.
- **tmp_name** (*str*) – Template spectrum filename.
- **tmp_reader** (*python routine*) – External routine that reads a model file and returns a 3D numpy array, where the first column corresponds to wavelengths, the second contains flux values, and the third the uncertainties on the flux.
- **search_mode** (*str, opt {'simplex', 'grid'}*) – How is the best fit template found? Simplex or grid search.
- **lambda_scal** (*float, optional*) – Wavelength where a first scaling will be performed between template and observed spectra. If not provided, the middle wavelength of the osberved spectra will be considered.
- **scale_range** (*tuple, opt*) – If grid search, this parameter should be provided as a tuple of 3 floats: lower limit, upper limit and step of the grid search for the scaling factor to be applied AFTER the first rough scaling (i.e. scale_range should always encompass 1).
- **ext_range** (*tuple or None, opt*) –
  - If None: differential extinction is not considered as a free parameter.

  - If a tuple: it should contain 2 floats (for simplex `search_mode`) or 3 floats (for grid search `search_mode`) corresponding to the lower limit, upper limit (and step for the grid search). For the simplex search, the lower and upper limits are used to set a chi square of infinity outside of the range.

- **dlbda_obs** (*numpy 1d ndarray or list, optional*) – Respective spectral channel width or FWHM of the photometric filters used for each point of the observed spectrum. This vector is used to infer which part(s) of a combined spectro+photometric spectrum should involve convolution+subsampling (model resolution higher than measurements), interpolation (the opposite), or convolution by the transmission curve of a photometric filter. If not provided, it will be inferred from the difference between consecutive lbda_obs points (i.e. inaccurate for a combined spectrum). It must be provided IF one wants to weigh each measurement based on the spectral resolution of each instrument (as in [OLO16]), through the `use_weights` argument.

- **instru_corr** (*numpy 2d ndarray, optional*) – Spectral correlation between post-processed images in which the spectrum is measured. It is specific to the instrument, PSF subtraction algorithm and radial separation of the companion from the central star. Can be computed using `special.spec_corr.spectral_correlation`. In case of a spectrum obtained with different instruments, it is recommended to construct the final spectral_correlation matrix with `special.spec_corr.combine_corrs`. If `instru_corr` is not provided, the uncertainties in each spectral channel will be considered independent. See [GRE16] for more details.

- **instru_res** (*float or list of floats/strings, optional*) – The mean instrumental resolving power(s) OR filter names. This is used to convolve the model spectrum. If several instruments are used, provide a list of resolving power values / filter names, one for each instrument used.

- **instru_idx** (*numpy 1d array, optional*) – 1d array containing an index representing each instrument used to obtain the spectrum, label them from 0 to the number of instruments (n_{ins}). Zero for points that don't correspond to any of the `instru_res` values provided, and i in $[1, n_{ins}]$ for points associated to instru_res[i-1]. This parameter must be provided if the spectrum consists of points obtained with different instruments.

- **use_weights** (*bool, optional*) – For the likelihood calculation, whether to weigh each point of the spectrum based on the spectral resolution or bandwidth of photometric filters used. Weights will be proportional to `dlbda_obs/lbda_obs` if `dlbda_obs` is provided, or set to 1 for all points otherwise.

- **filter_reader** (*python routine, optional*) – External routine that reads a filter file and returns a 2D numpy array, where the first column corresponds to wavelengths, and the second contains transmission values. Important: if not provided, but strings are detected in instru_res, the default file reader will be used. It assumes the following format for the files:

  - first row contains headers (titles of each column)
  - starting from 2nd row: 1st column: wavelength, 2nd col.: transmission
  - Unit of wavelength can be provided in parentheses of first header key name: e.g. "WL(AA)" for angstrom, "wavelength(mu)" for micrometer or "lambda(nm)" for nanometer. Note: only what is in parentheses matters for the units.

- **red_chi2** (*bool, optional*) – Whether to compute the reduced chi square. If False, considers chi^2.

- **remove_nan** (*bool, optional*) – Whether to remove NaN values from template spectrum BEFORE resampling to the wavelength sampling of the observed spectrum. Whether

it is set to True or False, a check is made just before chi^2 is calculated (after resampling), and only non-NaN values will be considered.

- **simplex_options** (*dict, optional*) – The `scipy.optimize.minimize` simplex (Nelder-Mead) options.
- **force_continue** (*bool, optional*) – In case of issue with the fit, whether to continue regardless (this may be useful in an uneven spectral library, where some templates have too few points for the fit to be performed).
- **verbose** (*str, optional*) – Whether to print more information when fit fails.
- **min_npts** (*int or None, optional*) – Iinimum number of (resampled) points to consider a template spectrum valid in the minimization search. A Nan value will be returned for chi if the condition is not met.
- ****kwargs** (*optional*) – Other optional arguments to the `scipy.optimize.minimize` function.

Returns

- **best_chi** (*float*) – goodness of fit scored by the template
- *best_scal* – best-fit scaling factor for the considered template
- *best_ext* – best-fit optical extinction for the considered template

---

Note: If several filter filenames are provided in `instru_res`, the filter files must all have the same format and wavelength units (for reading by the same `filter_reader` snippet or default function).

15.11 special.utils_mcmc module

Module with utility functions to the MCMC (`emcee`) sampling for parameter estimation.

`special.utils_mcmc.`**autocorr_test**(*chain*)

Function to measure the auto-correlation 'timescale' of the chain, normalized by the length of the whole chain up to that point. This metrics can then be used to check if the chain has likely converged. More details here: https://emcee.readthedocs.io/en/stable/tutorials/autocorr/

Parameters
chain (*numpy.array*) – The numpy.array on which the auto-correlation is calculated.

Returns
tau/N – The normalized auto-correlation timescale.

Return type
float

`special.utils_mcmc.`**gelman_rubin**(*x*)

Determine the Gelman-Rubin hat{R} statistical test between Markov chains.

Parameters
x (*numpy.array*) – The numpy.array on which the Gelman-Rubin test is applied. This array should contain at least 2 set of data, i.e. x.shape >= (2,).

Returns
out – The Gelman-Rubin hat{R}.

Return type
float

Example

```
>>> x1 = np.random.normal(0.0,1.0,(1,100))
>>> x2 = np.random.normal(0.1,1.3,(1,100))
>>> x = np.vstack((x1,x2))
>>> gelman_rubin(x)
1.0366629898991262
>>> gelman_rubin(np.vstack((x1,x1)))
0.99
```

special.utils_mcmc.**gelman_rubin_from_chain**(*chain*, *burnin*)

Pack the MCMC chain and determine the Gelman-Rubin hat{R} statistical test. In other words, two sub-sets are extracted from the chain (burnin parts are taken into account) and the Gelman-Rubin statistical test is performed.

Parameters

- **chain** (*numpy.array*) – The MCMC chain with the shape walkers x steps x model_parameters
- **burnin** (*float in [0,1]*) – The fraction of a walker which is discarded.

Returns
out – The Gelman-Rubin hat{R}.

Return type
float

15.12 special.utils_nested module

Module with utility functions to the nested sampling for parameter estimation.

special.utils_nested.**un_burning**(*res*, *logger=None*)

Automatic burning of UltraNest chain based on cumulated sum of weights (as implemented in UltraNest's cornerplot).

Note: this function is necessary to be able to make corner plots showing units after best estimates, as ultranest's cornerplots does not feature that option and does burning+corner plot together.

Parameters
res (*UltraNest result object*) – The UltraNest result.

Returns
burned_res – The burned UltraNest chain and associated weights

Return type
tuple of 2 numpy nd array

15.13 special.utils_spec module

Utility functions for spectral fitting.

special.utils_spec.**akaike**(*LnL*, *k*)

Computes the Akaike Information Criterion: 2k-2ln(L), where k is the number of estimated parameters in the model and LnL is the max ln-likelihood for the model.

Parameters

- **LnL** (*float*) – Max ln-likelihood for the considered model.
- **k** (*int*) – Number of estimated parameters in the model.

Returns
aic – Akaike Information Criterion

Return type
float

special.utils_spec.**blackbody**(*lbda*, *T*)

Planck function. Returns specific intensity for an input wavelength vector lbda (in micrometers) and a given input temperature.

Parameters

- **lbda** (*numpy array*) – 1d numpy array corresponding to the wavelengths (in microns) for the desired output specific intensities.
- **T** (*float*) – Temperature

Returns
B_lambda – Specific intensity corresponding to the Planck function.

Return type
float

special.utils_spec.**convert_F_units**(*F*, *lbda*, *in_unit='cgs'*, *out_unit='si'*)

Function to convert Flux density between [ergs s-1 cm-2 um-1], [W m-2 um-1] and [Jy].

Parameters

- **F** (*float or 1d array*) – Flux
- **lbda** (*float or 1d array*) – Wavelength of the flux (in um)
- **in_unit** (*str, opt, {"si", "cgs", "jy", "cgsA"}*) – Input flux units. 'si': W/m^2/mu; 'cgs': ergs/s/cm^2/mu 'jy': janskys 'cgsA': erg/s/cm^2/AA
- **out_unit** (*str, opt {"si", "cgs", "jy"}*) – Output flux units.

Return type
Flux in output units.

special.utils_spec.**convert_F_vs_mag**(*value*, *F_0=None*, *band='H'*, *system='Johnson'*, *conversion='to_mag'*)

Function to convert Flux density (in Jy) to magnitude in a given band, or the opposite.

Sources for zero points:

- TOKUNAGA chapter on IR astronomy (from Cohen 1992)
- UKIRT webpage: (http://www.jach.hawaii.edu/UKIRT/astronomy/calib/phot_cal/conver.html)

- van der Bliek et al. 1996 (ESO standard stars)

Parameters

- **value** (*float*) – Flux or magnitude to be converted.
- **F_0** (*float, opt*) – Zero-point flux. If provided will take precedence over band.
- **band** (*str, opt*) – Band of the given flux or magnitude. Choice between: {'U','B','V', 'R', 'I', 'J', 'H', 'K', "L", "L'", 'M', 'N', 'O'} (but not for all band systems).
- **system** (*str, opt*) – Band system. Choice between: {'Johnson;,'2MASS', 'UKIRT', 'ESO'}
- **conversion** (*str, opt*) – In which sense to convert: flux to mag ('to_mag') or mag to flux ('to_flux')

Return type
Converted flux or magnitude.

special.utils_spec.**extinction**(*lbda*, *AV*, *RV=3.1*)

Calculates the A(lambda) extinction for a given combination of A_V and R_V. If R_V is not provided, assumes an ISM value of R_V=3.1 Uses the Cardelli et al. (1989) empirical formulas.

Parameters

- **lbda** (*1d np.ndarray*) – Array with the wavelengths (um) for which the extinction is calculated.
- **AV** (*float*) – Extinction (mag) in the V band.
- **RV** (*float, opt*) – Reddening in the V band: R_V = A_V / E(B-V)

Returns
Albda – Extinction (mag) at wavelengths lbda.

Return type
1d np.ndarray

special.utils_spec.**find_nearest**(*array*, *value*, *output='index'*, *constraint=None*, *n=1*)

Function to find the indices, and optionally the values, of an array's n closest elements to a certain value.

Parameters

- **array** (*1d numpy array or list*) – Array in which to check the closest element to value.
- **value** (*float*) – Value for which the algorithm searches for the n closest elements in the array.
- **output** (*str, opt {'index','value','both' }*) – Set what is returned
- **constraint** (*str, opt {None, 'ceil', 'floor'}*) – If not None, will check for the closest element larger than value (ceil) or closest element smaller than value (floor).
- **n** (*int, opt*) – Number of elements to be returned, sorted by proximity to the values. Default: only the closest value is returned.

Returns

- *Either* – (output='index'): index/indices of the closest n value(s) in the array; (output='value'): the closest n value(s) in the array, (output='both'): closest value(s) and index/-ices, respectively.

- *By default, only returns the index/indices.*
- **Possible constraints** (*'ceil', 'floor', None ("ceil" will return the closest*)
- *element with a value greater than 'value', "floor" the opposite)*

special.utils_spec.**inject_em_line**(*wl*, *flux*, *lbda*, *spec*, *width=None*, *height=0.1*, *em=True*)

Injects an emission (or absorption) line in a spectrum.

Parameters

- **wl** (*float*) – Wavelength of the line
- **flux** (*float*) – Flux of the line to be injected
- **lbda** (*1d np.ndarray*) – Array with the wavelengths (um) of the input spectrum.
- **spec** (*1d np.ndarray*) – Input spectrum fluxes
- **width** (*float, opt*) – Full width of the line in mu (see also height). The line will be injected assuming a gaussian profile. If not provided, the width will be set to the 'equivalent width' of the line.
- **height** (*float, opt*) – Ratio to peak where the line width is considered. E.g. if height=10%, the width will be the full width at 10% maximum.
- **em** (*bool, opt*) – Whether emission (True) or absorption (False) line.

Returns

spec – Spectrum with the injected line

Return type

1d np.ndarray

special.utils_spec.**mj_from_rj_and_logg**(*rp*, *logg*)

Estimates a planet mass in Jupiter mass for a given radius in Jupiter radius and the log of the surface gravity.

Parameters

- **rp** (*float*) – Planet radius in Jupiter radii
- **logg** (*float*) – Log of the surface gravity

Returns

mj – Planet mass in Jupiter masses

Return type

float

special.utils_spec.**nrefrac**(*wavelength*, *density=1.0*)

Calculates refractive index of air from Cauchy formula. For comparisong to measurements from the ground, the wavelenghts of model spectra must be slightly shifted using: lbda_shift = lbda_model/(1+(nrefrac*1e-6))

Input: wavelength in Angstrom, Returns N = (n-1) * 1.e6. Credit: France Allard.

Parameters

- **wavelength** (*numpy array*) – 1d numpy array corresponding to the wavelengths of the input spectrum in Angstrom
- **density** (*float*) – density of air in amagat (relative to STP, e.g. ~10% decrease per 1000m above sea level).

Returns

N – Refractive index

Return type
float

15.14 Module contents

`special` has helping functions for the analysis of (low-res) spectra, including:

- fitting of input spectra to models and templates;
- mcmc sampling of model parameter space;
- nested sampling of model parameter space;
- best fit search within a template library;
- utility functions for the spectral fit.

CHAPTER

SIXTEEN

API

- genindex
- modindex
- search

BIBLIOGRAPHY

[OLO16] Olofsson et al. 2016
Azimuthal asymmetries in the debris disk around HD 61005. A massive collision of planetesimals?
Astronomy & Astrophysics, Volume 591, p. 108
https://arxiv.org/abs/1601.07861

[GRE16] Greco & Brandt 2016
The Measurement, Treatment, and Impact of Spectral Covariance and Bayesian Priors in Integral-field Spectroscopy of Exoplanets
The Astrophysical Journal, Volume 833, Issue 1, p. 134
https://arxiv.org/abs/1602.00691

[FOR13] Foreman-Mackey et al. 2013
emcee: The MCMC Hammer
PASP, Volume 125, Issue 925, p. 306
https://arxiv.org/abs/1202.3665

[FOR19] Foreman-Mackey et al. 2019
emcee v3: A Python ensemble sampling toolkit for affine-invariant MCMC
JOSS, Volume 4, Issue 43, p. 1864
https://arxiv.org/abs/1911.07688

[GOO10] Goodman & Weare 2010
Ensemble samplers with affine invariance
Comm. App. Math. Comp. Sci., Vol. 5, Issue 1, pp. 65-80.
https://ui.adsabs.harvard.edu/abs/2010CAMCS…5…65G

[nes13] Barbary 2013
nestle
GitHub
https://github.com/kbarbary/nestle

[SKI04] Skilling 2004
Bayesian Inference and Maximum Entropy Methods in Science and Engineering: 24th International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering
American Institute of Physics Conference Series, Volume 735, pp. 395-405
https://ui.adsabs.harvard.edu/abs/2004AIPC..735..395S

[MUK06] Mukherjee et al. 2006
A Nested Sampling Algorithm for Cosmological Model Selection
ApJL, Volume 638, Issue 2, pp. 51-54

https://arxiv.org/abs/astro-ph/0508461

[FER09] Feroz et al. 2009
MULTINEST: an efficient and robust Bayesian inference tool for cosmology and particle physics
MNRAS, Volume 398, Issue 4, pp. 1601-1614
https://arxiv.org/abs/0809.3437

[BUC21] Buchner 2021
UltraNest - a robust, general purpose Bayesian inference engine
JOSS, Volume 6, Issue 60, p. 3001
https://arxiv.org/abs/2101.09604

[GOR03] Gorlova et al. 2003
Gravity Indicators in the Near-Infrared Spectra of Brown Dwarfs
The Astrophysical Journal, Volume 593, Issue 1, pp. 1074-1092
https://arxiv.org/abs/astro-ph/0305147

[SLE04] Slesnick et al. 2004
The Spectroscopically Determined Substellar Mass Function of the Orion Nebula Cluster
The Astrophysical Journal, Volume 610, Issue 1, pp. 1045-1063
https://arxiv.org/abs/astro-ph/0404292

[ALL07] Allers et al. 2007
Characterizing Young Brown Dwarfs Using Low-Resolution Near-Infrared Spectra
The Astrophysical Journal, Volume 657, Issue 1, pp. 511-520
https://arxiv.org/abs/astro-ph/0611408

PYTHON MODULE INDEX

S

INDEX

S

T

U

W